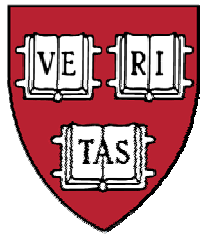
Microarchitecture-Level Power-Performance Simulators: Modeling, Validation, and Impact on Design

Zhigang Hu, David Brooks, Victor Zyuban, Pradip Bose

IBM Research

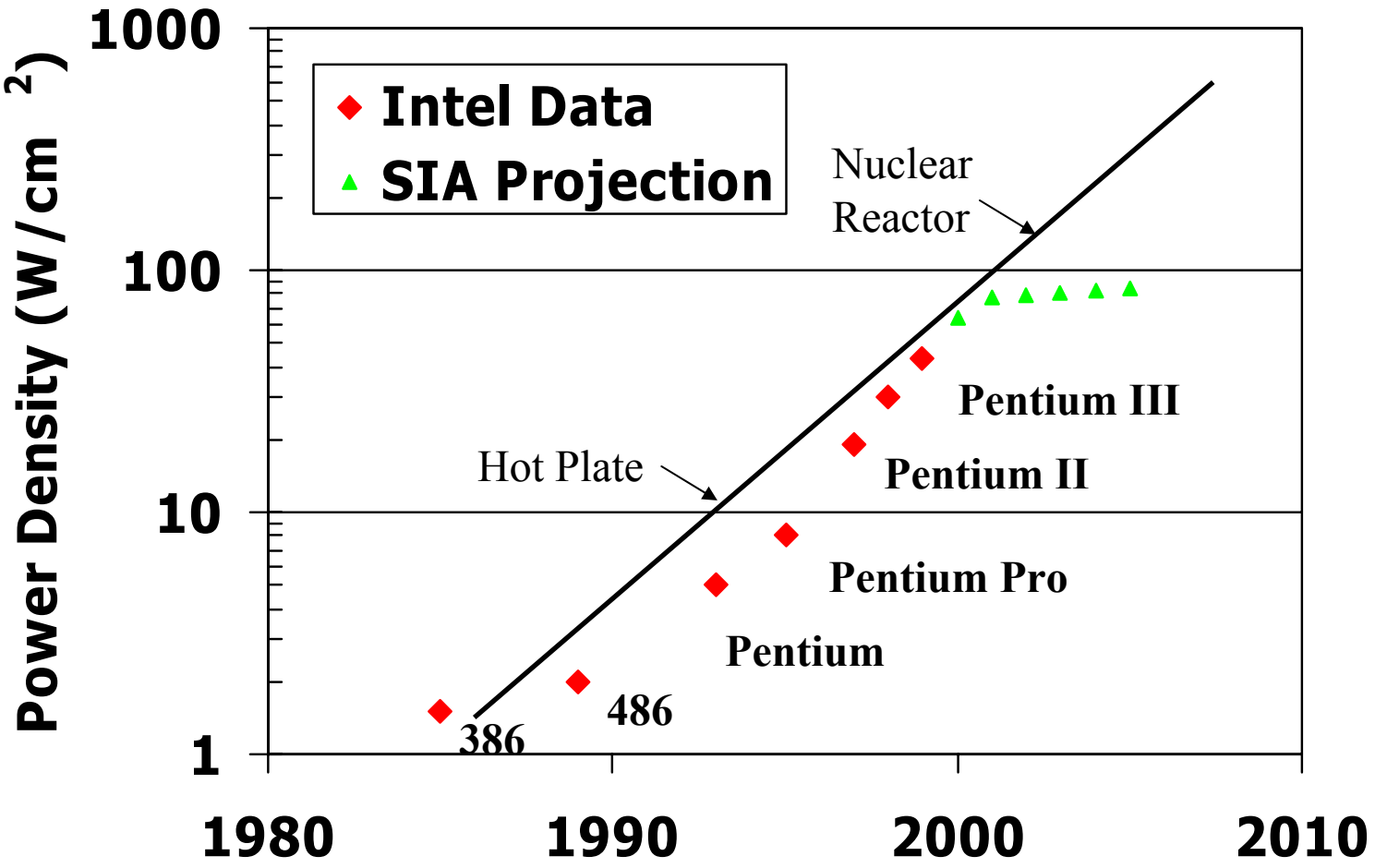
Harvard University



Tutorial Outline

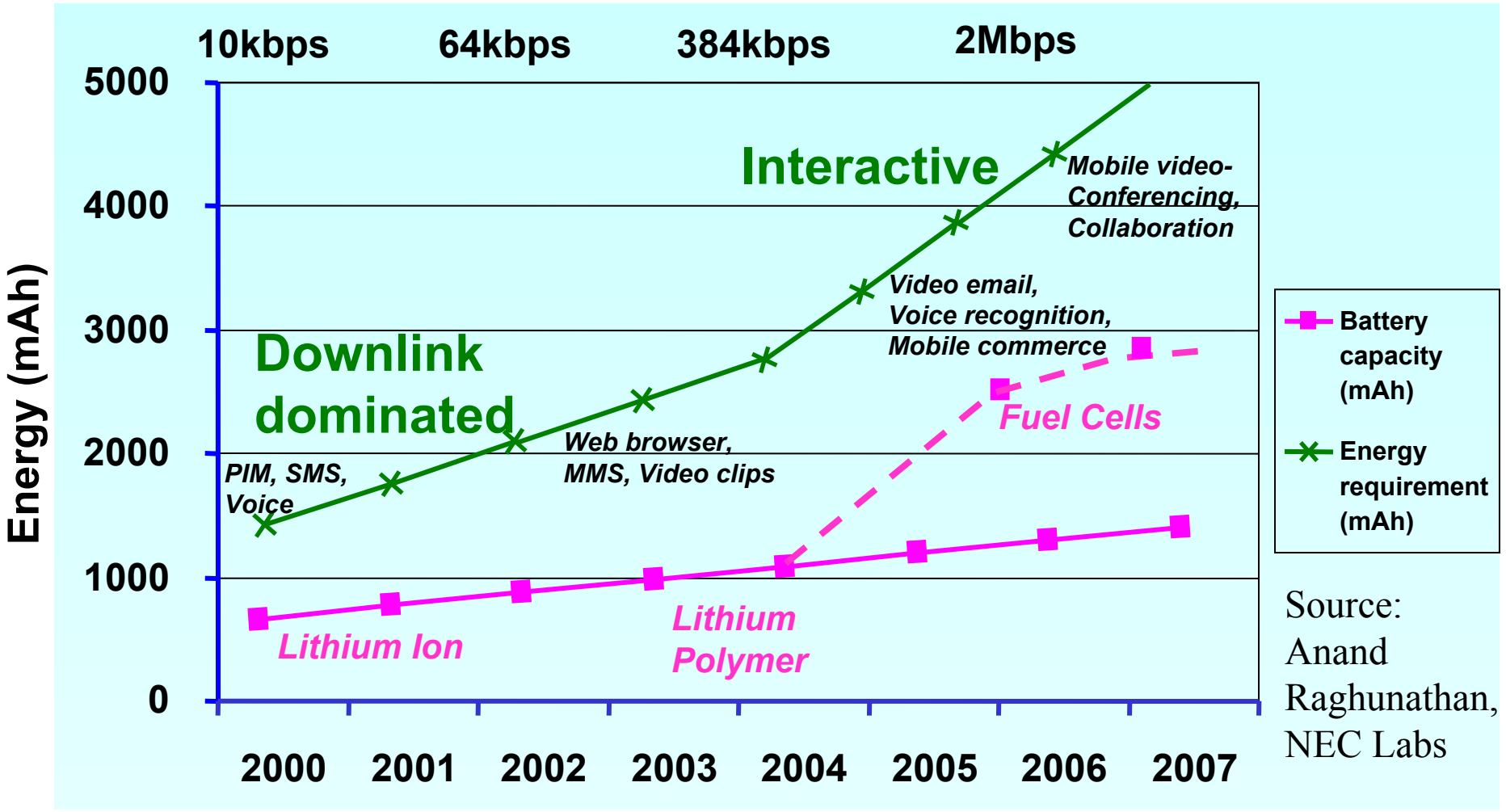
8:00-8:15	Introduction and Motivation Basics of Performance Modeling - Turandot performance simulation infrastructure Architectural Power Modeling - PowerTimer extensions to Turandot - Power-Performance Efficiency Metrics Case Studies and Examples - Optimal Power-Performance Pipeline Depth Validation and Calibration Efforts Future challenges and Discussion Bibliography

Power Dissipation Trends



The Battery Gap

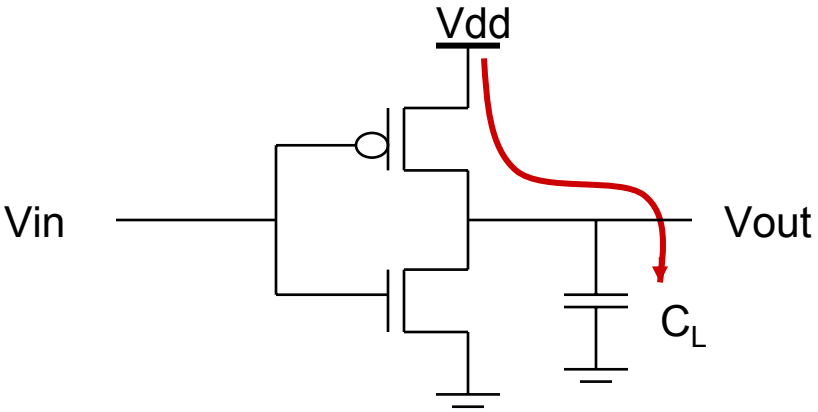
Diverging Gap Between Actual Battery Capacities and Energy Needs



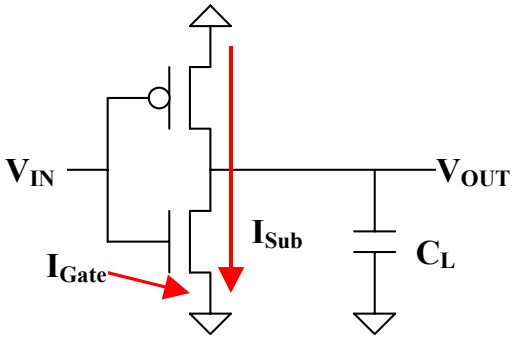
Source: Anand Raghunathan, NEC Labs

Power Issues

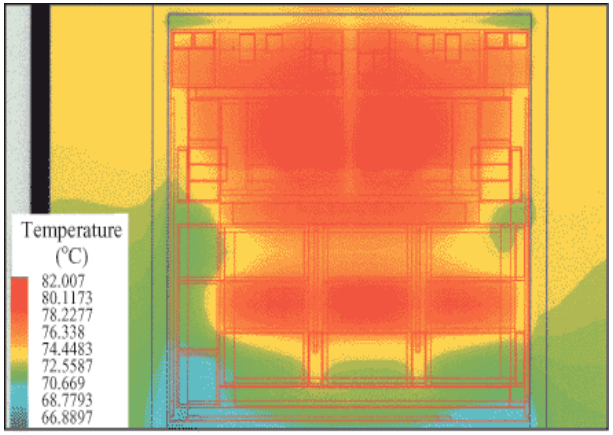
Capacitive (Dynamic) Power



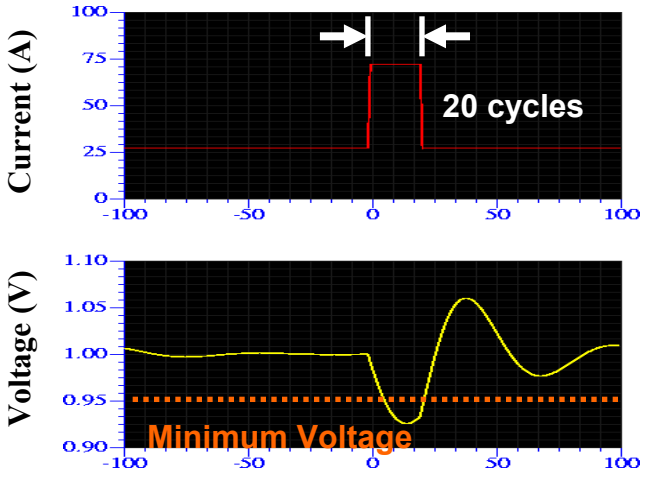
Static (Leakage) Power



Temperature



Di/Dt (Vdd/Gnd Bounce)



Application Areas for Power-Aware Computing

Why architecture/system level?

- Many architectural/system decisions have huge impact on power and performance
- Often need feedback at the early-stage of a design
  - Pre-RTL, pre-circuit analysis
- Run-time, system-level feedback control
  - Application/dynamic run-time characteristics allow dynamic scaling for power reduction
  - Perhaps power, temperature, and voltage sensor to guide throttling for worst-case situations

What architects need from lower levels…

- Architects need abstract models on many levels…
  - Static speed-power knobs for structures
    - Parameterized models for HW structures
    - Impact of implementation choices
  - Given cycle-level power estimates (power vs. time)
    - Chip temperature models
    - Chip di/dt models
- Hardware hooks
  - Dynamic speed-power knobs for structures
    - Clock gating, Vdd-scaling, Vdd-gating
    - Need to understand costs of these knobs
  - On-chip sensors to measure power, temperature, voltage deltas

Tutorial Outline

8:15-9:00	Introduction and Motivation Basics of Performance Modeling - Turandot performance simulation infrastructure Architectural Power Modeling - PowerTimer extensions to Turandot - Power-Performance Efficiency Metrics Case Studies and Examples - Optimal Power-Performance Pipeline Depth Validation and Calibration Efforts Future challenges and Discussion Bibliography

A Developer's Guide to Turandot/PowerTimer

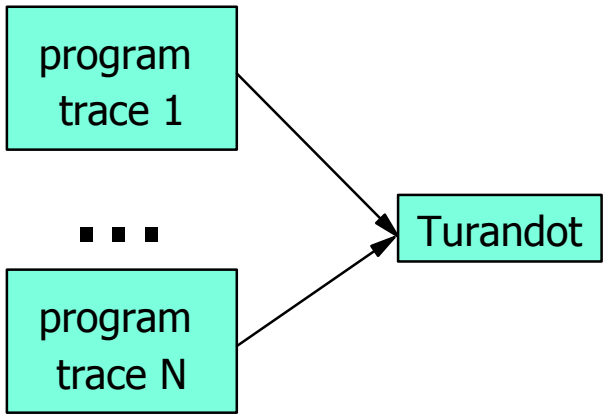
- Acknowledgments: J-D Wellman, Jaime Moreno, and other IBMers in the original Turandot/MET development team

Processor Simulator: An Overview

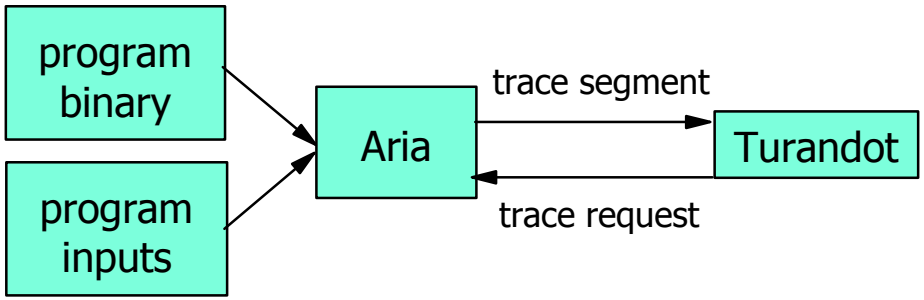
- Processor simulator: a tool that emulates the behavior of a real processor
  - Software-based:
    - Concept phase: C/C++/System C
    - Design phase: VHDL
  - Hardware-based:
    - FPGA
- Simulators are used for:
  - Workload characterization
  - Performance / power target projection
  - Compiler tuning
  - Design space exploration and tradeoff evaluation
  - Testing / debugging/ validation
- Existing simulators
  - Academia simulators: SimpleScalar, RSIM, SMTSIM, etc.
  - Industry simulators:
    - Concept phase
    - Product phase

Turandot/PowerTimer Overview

- An out-of-order superscalar processor model for the PowerPC architecture
  - Cycle-accurate, cycle-based
  - Initial version developed by a group of researchers at IBM T.J. Watson
  - Power4-like machine configuration by default
  - Other configurations attainable through compile-time parameters
- Performance model validated against Power4 preRTL model
- Power model added in summer 2000
  - Based on circuit simulation of Power4-like circuits
- Supporting trace-driven and execution-driven modes
  - Trace-driven mode now supports SMT, and is portable to AIX/Linux/Cygwin
  - Interpretation-based execution mode is underway



trace-driven mode

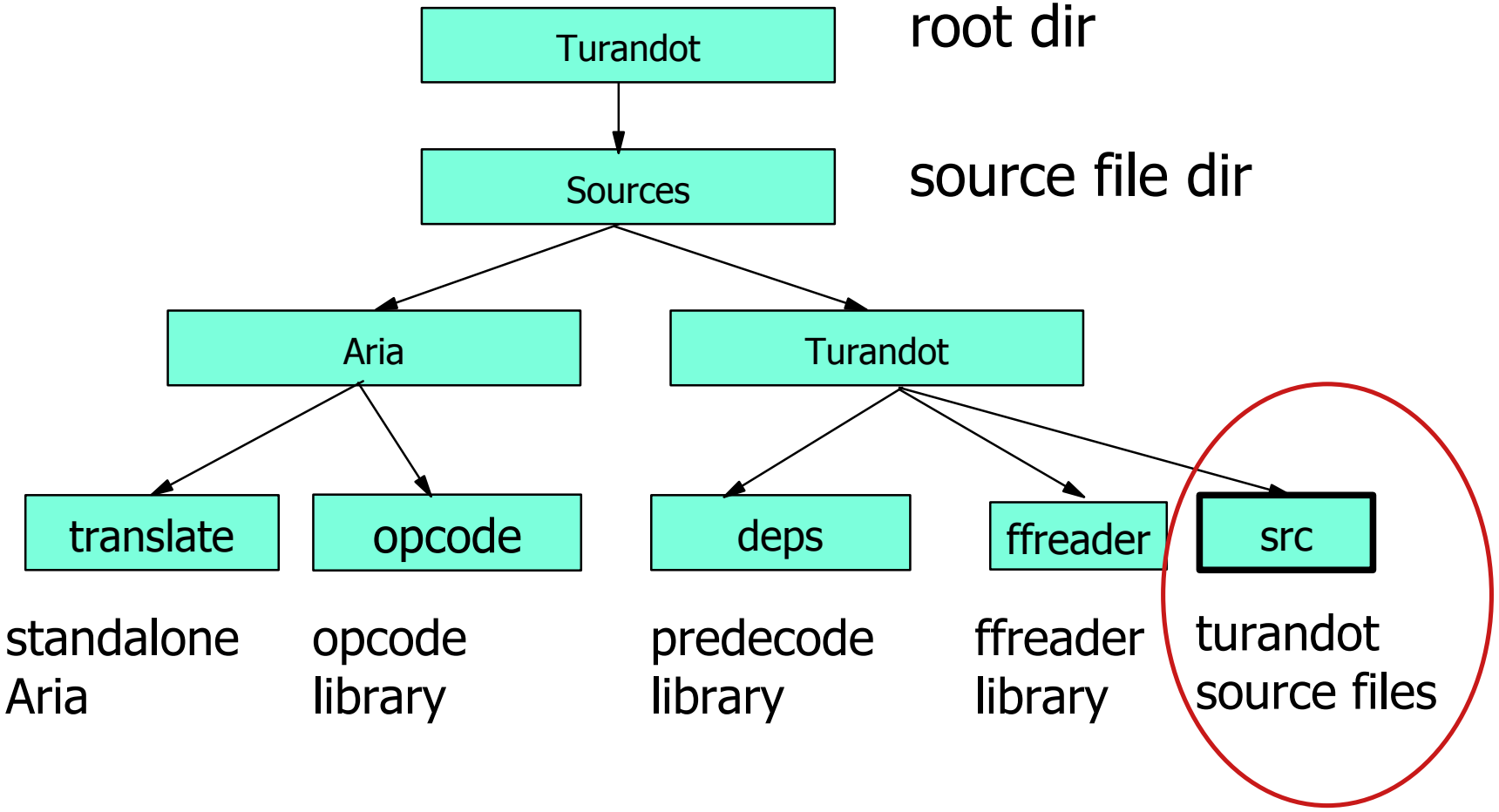


execution-driven mode

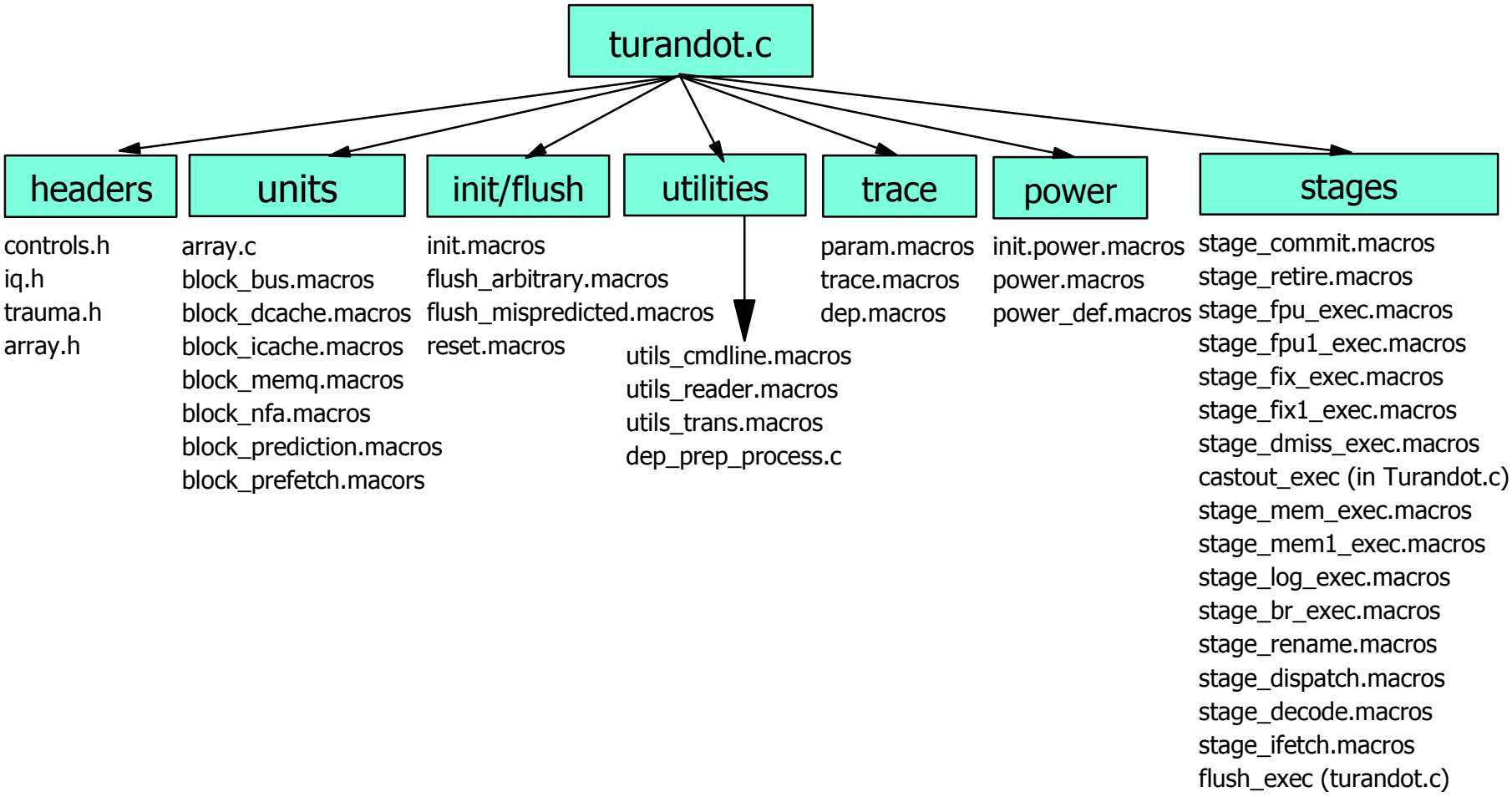
Disclaimer

1. Power4-like != Power4
2. Simulator implementation != real hardware implementation

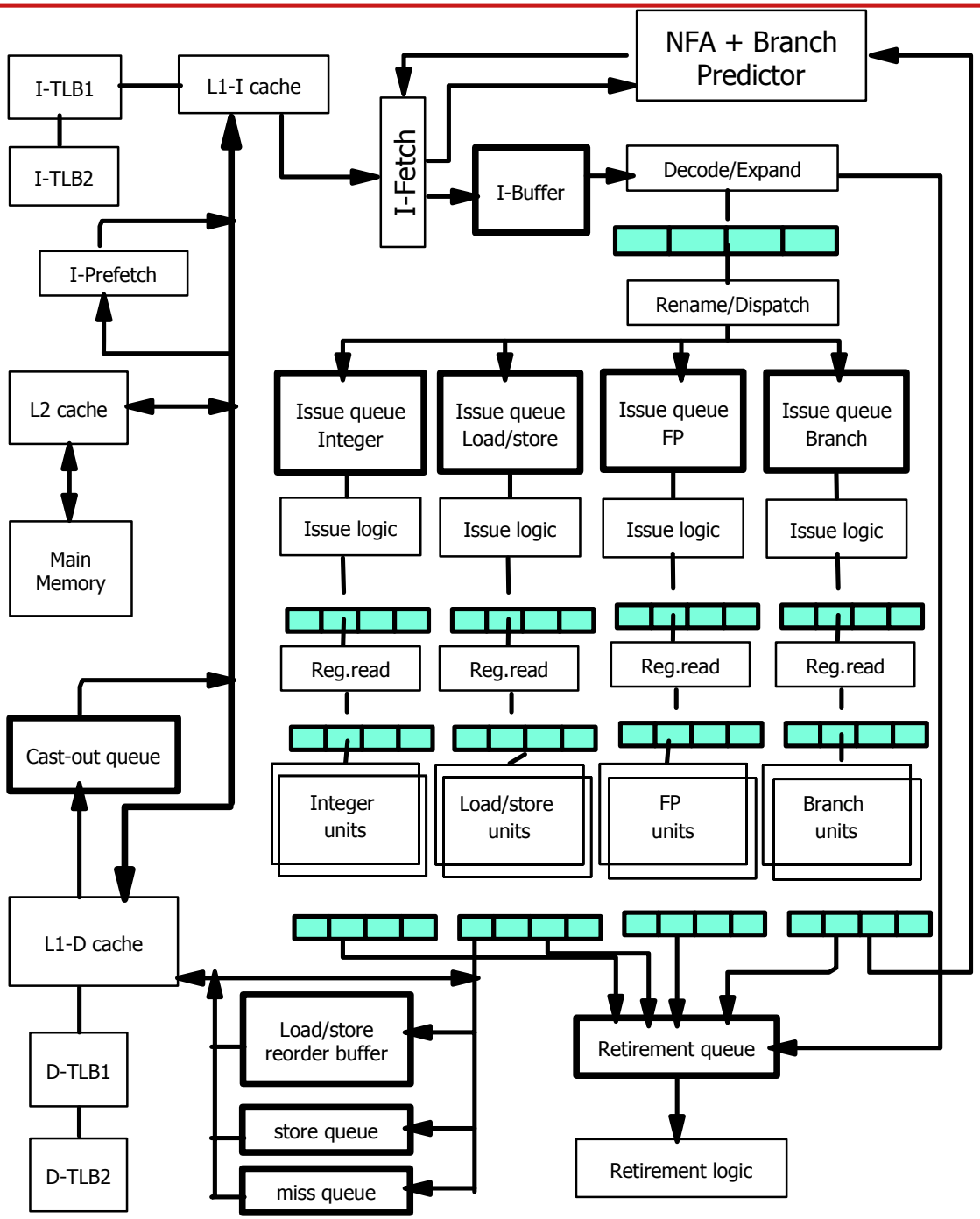
Source File Organization

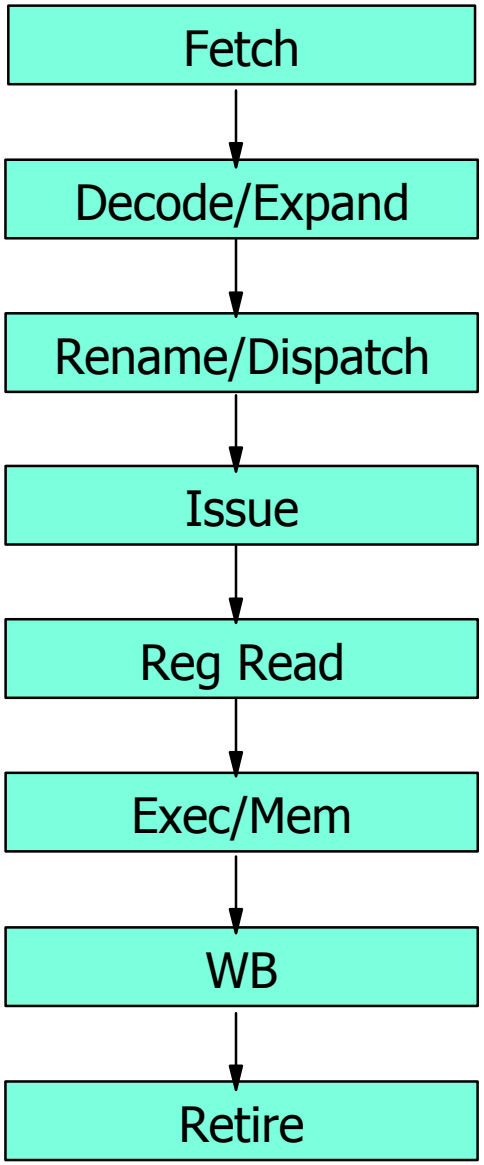


Turandot Source Files



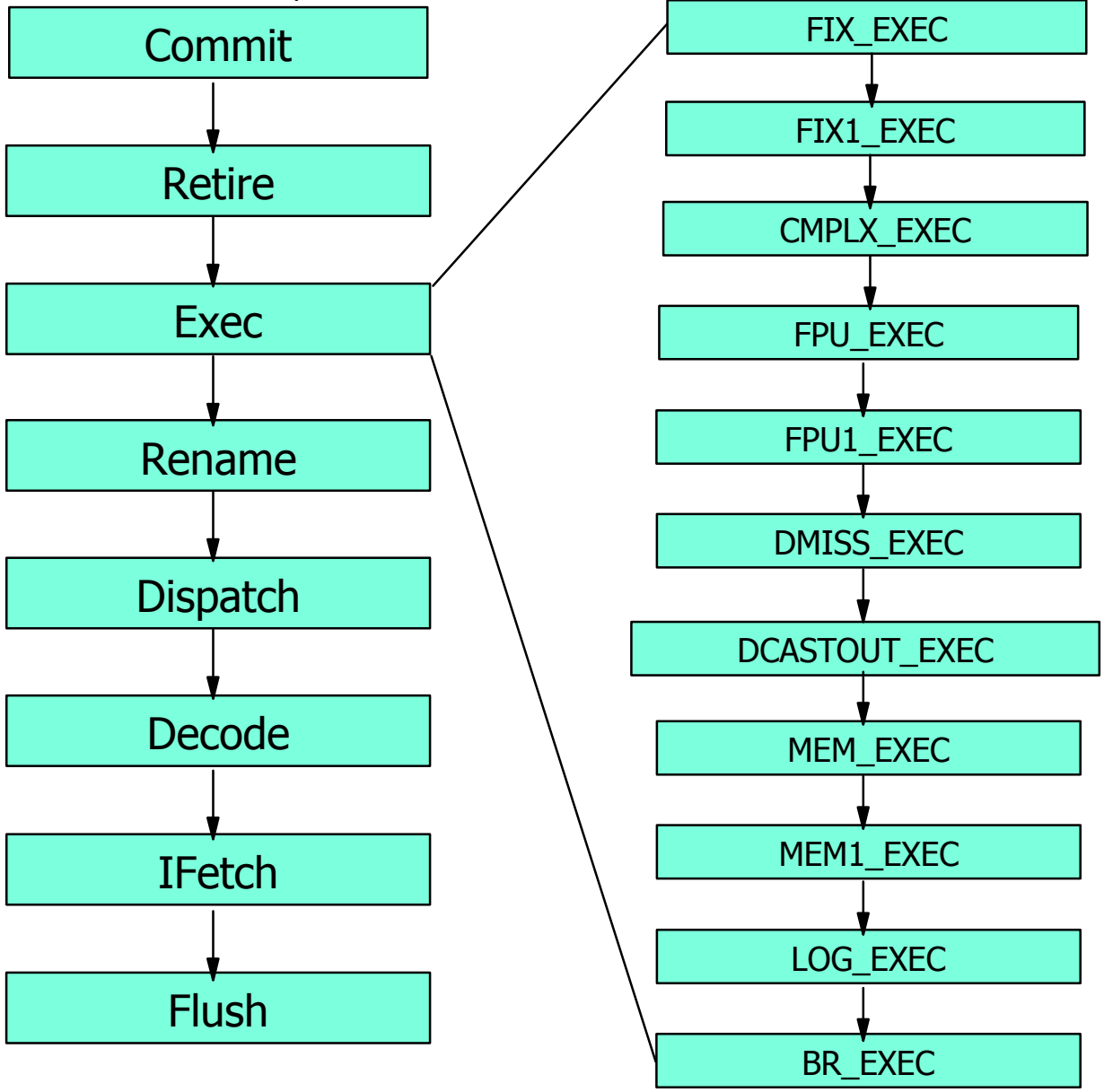
Turandot Simulation Framework





Simulation Flow: Reverse Pipeline Order

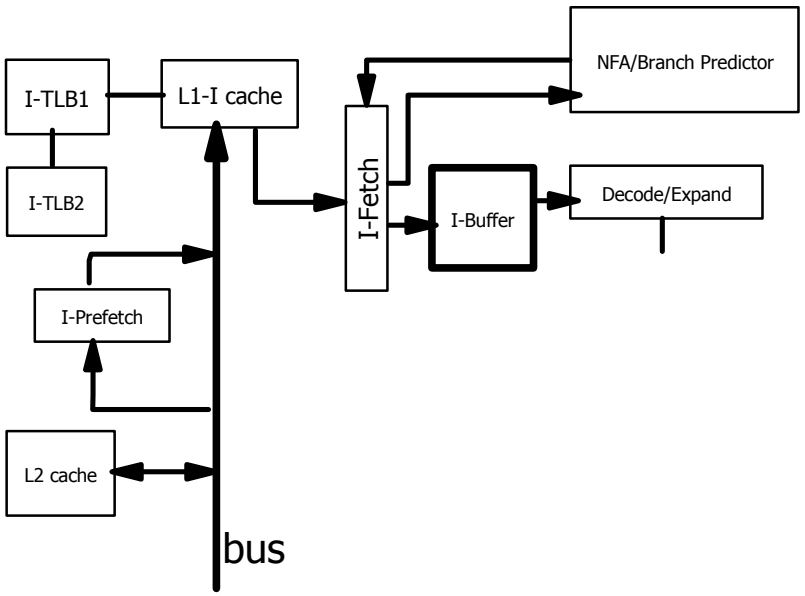
stores are committed to cache/memory COMMIT_STORES_DELAY cycles after retire



1. If store, remove from storeq
2. If load, remove from reorderq, check if there is a load/store conflict. If yes, flush the pipeline (reset.macros)
3. Update branch history
4. Remove instruction from retireq

1. Rename
   - I. Check if enough rename registers available, if not, stall until available.
   - II. Rename architectural registers to physical registers.
   - III. If the instruction is a mispredicted branch instruction, check if all operands are ready. If yes, resolve the branch and start fetching from the right path from next cycle.
   - IV. Note: registers in different class are renamed separately.
2. Dispatch
   1. Place renamed IOPs into the corresponding issue queue. If a given operation can not be placed in the issue queue (i.e. the queue is full), stall the stage until available.

Fetch Stage

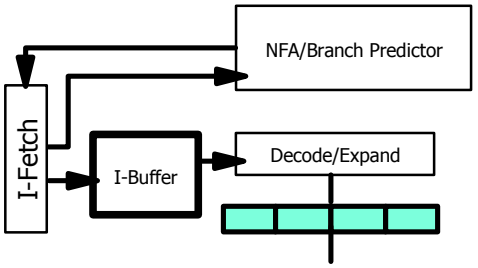


stage_ifetch.macros
    main fetch logic
array.c/h
    arrays: caches, counter prediction table, NFA, etc.

unit definitions:
    block_icache.macros
    block_bus.macros
    block_nfa.macros
    block_prefetch.macros
    block_prediction.macros

1. If a mispredicted branch is resolved (therefore ifetch has been on the mispredicted path), then revert back to the true taken path, flush the pipeline, and stall ifetch for a number of cycles.
2. If ifetch is stalled for some reason, check whether the reason has been resolved. If so, resume ifetch from next cycle.
3. Stall ifetch if I-Buffer is full, or no more fetch blocks are allowed, or no more inflight insns are allowed.
4. Fill the trace reader buffer. Stall ifetch if no instruction is available due to (1). no trace on the path (2). end of trace.
5. Use address of the first insn in this fetch block to:
    1. Check ITLB1 / ITLB2. If miss, stall ifetch for a number of cycles according to the miss type.
    2. Check L1 ICACHE / IPrefetch / L2 ICACHE. If miss, stall ifetch and charge appropriate miss penalties.
    3. Lookup NFA for next fetch address.
6. For each insn in current fetch block:
    1. Decode (see process_iword), expand into IOPs (internal insns), and insert IOPs into I-Buffer.
    2. If branch, perform branch prediction.
7. Update NFA.

Decode/Expand Stage



1. Expand instructions into IOPs and insert them into I-Buffer, in program order. (This code is in stage_ifetch.macros but logically it belongs to decode stage)
2. Handle millicode instructions (insns that expand to more than two IOPs), such as string ops. Stall decode if necessary.
3. Form instruction groups according to Power4 grouping rules (see IBM JR&D Power4 paper).

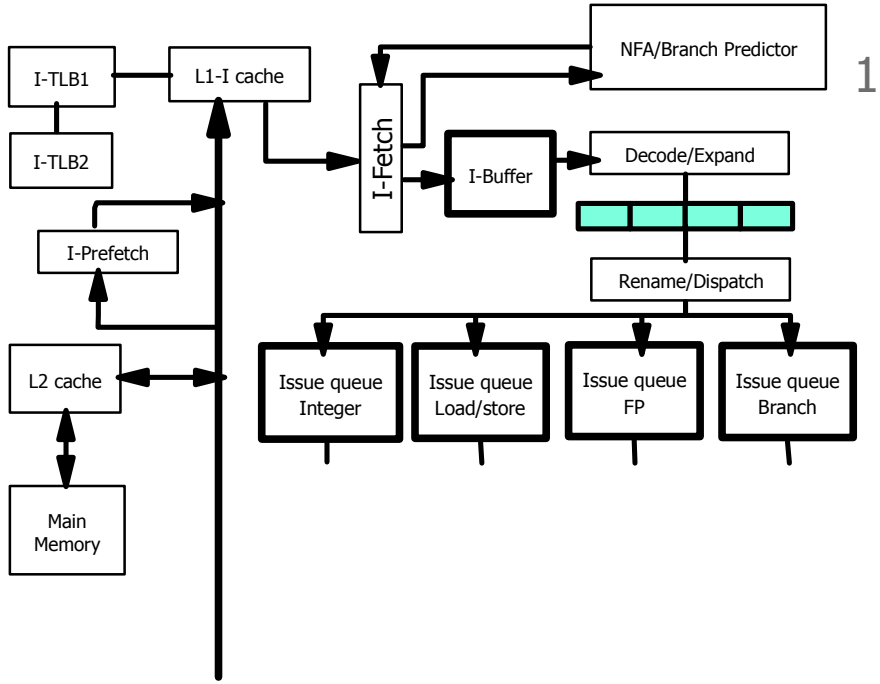
stage_ifetch.macros

expand instructions

stage_decode.macros

millicode handling and instruction group formation

Rename/Dispatch Stage



stage_rename.macros

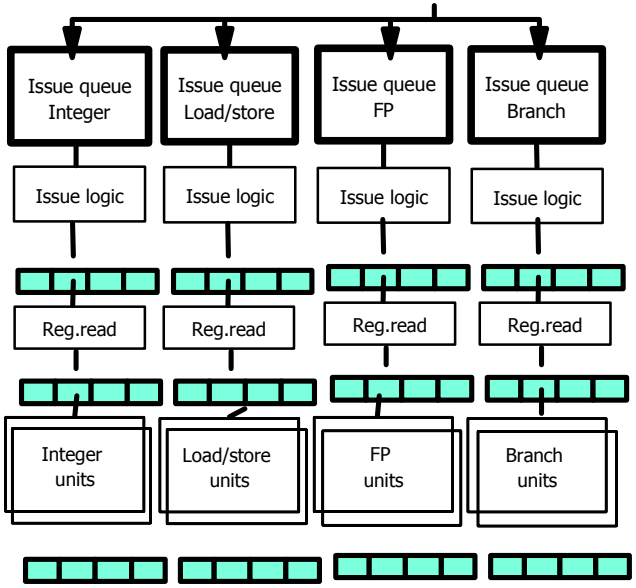
rename instructions

stage_dispatch.macros

dispatch instructions into corresponding issue queues.

1. Rename
   I. Check if enough physical registers available, if not, stall until available.
   II. Rename operands to physical registers, and allocate physical registers for each result.
   III. If the insn is a branch, check if all operands are ready. If so, resolve the branch and start fetching from the true path from next cycle.
   IV. If the insn is a mispredicted branch, checkpoint rename map for later recovery.
2. Dispatch
   1. For load/store, allocate reorderq/storeq slots if necessary. If no slot is available, stall dispatch.
   2. Dispatch IOPs into corresponding issue queues. Stall dispatch if no issue queue slot is available.

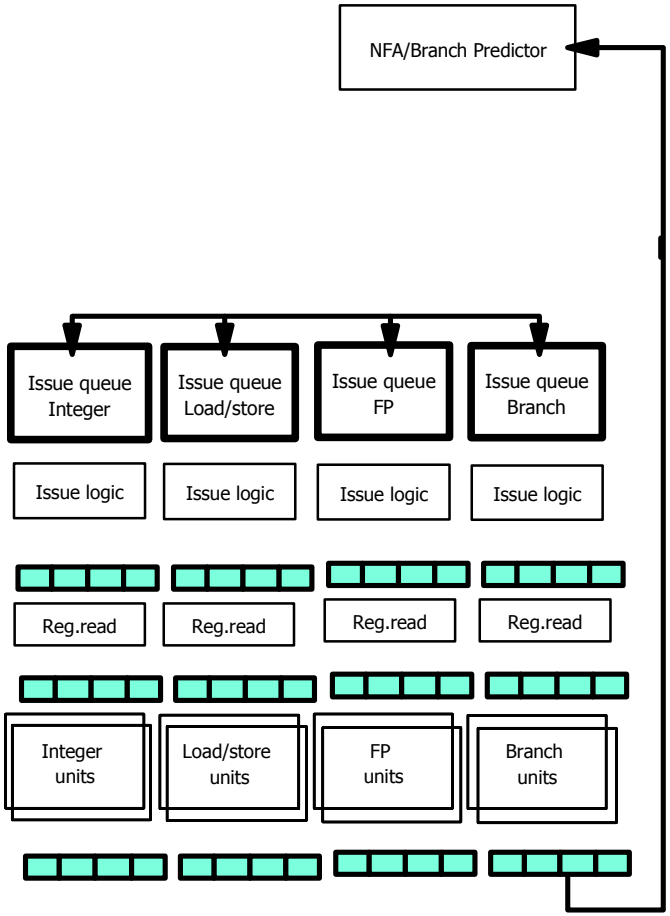
Issue/Execution Stage: FXU, FPU, LOG, CMPLX



1. Check if any non-pipelined instruction is in progress, if so, stall the pipeline
2. Issue ready insns in oldest-first order
3. Set result to be available after a number of cycles depending on the instruction latency
4. Remove insn from issue queue.

stage_fix_exec.macros/stage_fix1_exec.macros
    fix point instruction execution
stage_fpu_exec.macros/stage_fpu1_exec.macros
    floating point instruction execution
stage_log_exec.macros
stage_cmplx_exec.macros
    logic and complex instruction execution

Issue/Execution Stage: BR

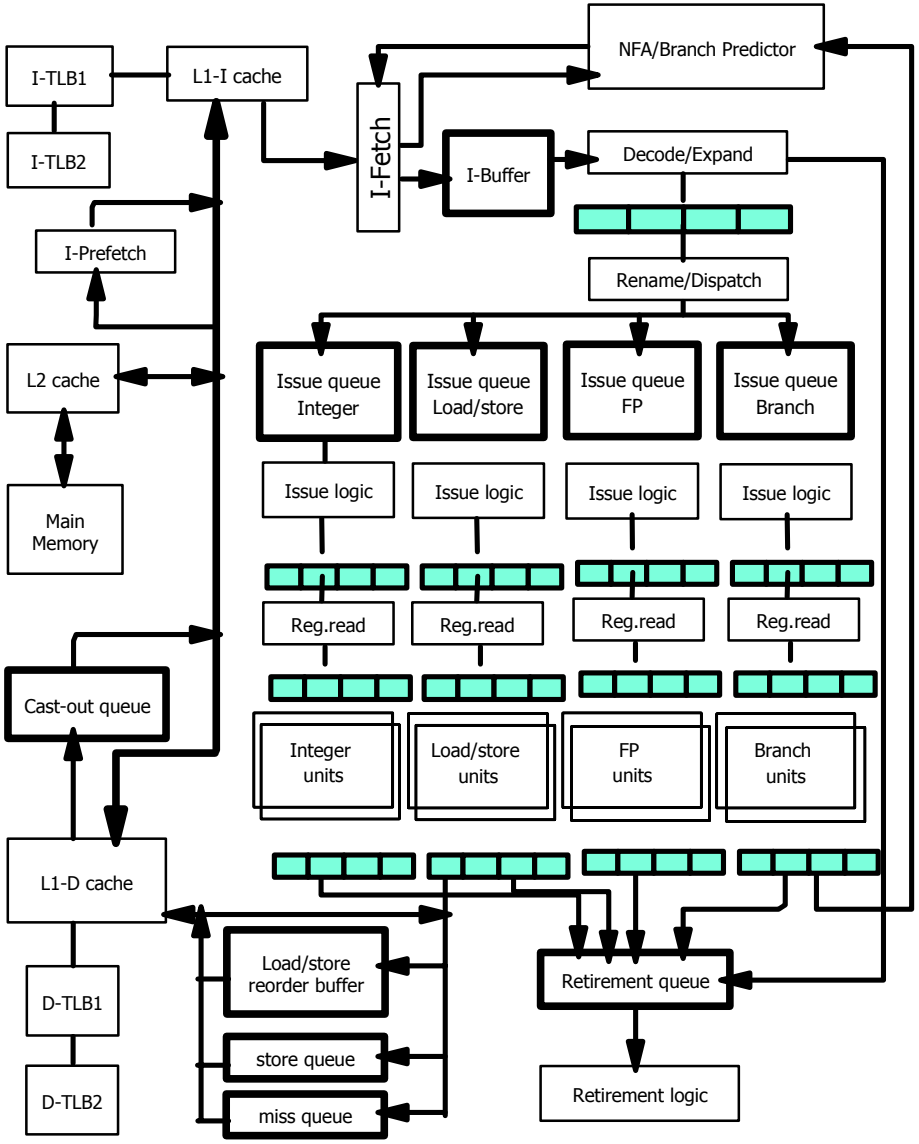


stage_br_exec.macros

branch execution

1. (if INORDER_BRANCHES), check if there are memory ops before this branch. If so, stall.
2. If operands are not ready, exit.
3. Collect branch stats.
4. If branch is mispredicted, perform some bookkeeping for preparation of pipeline flush.
5. Remove branch from branch issue queue.

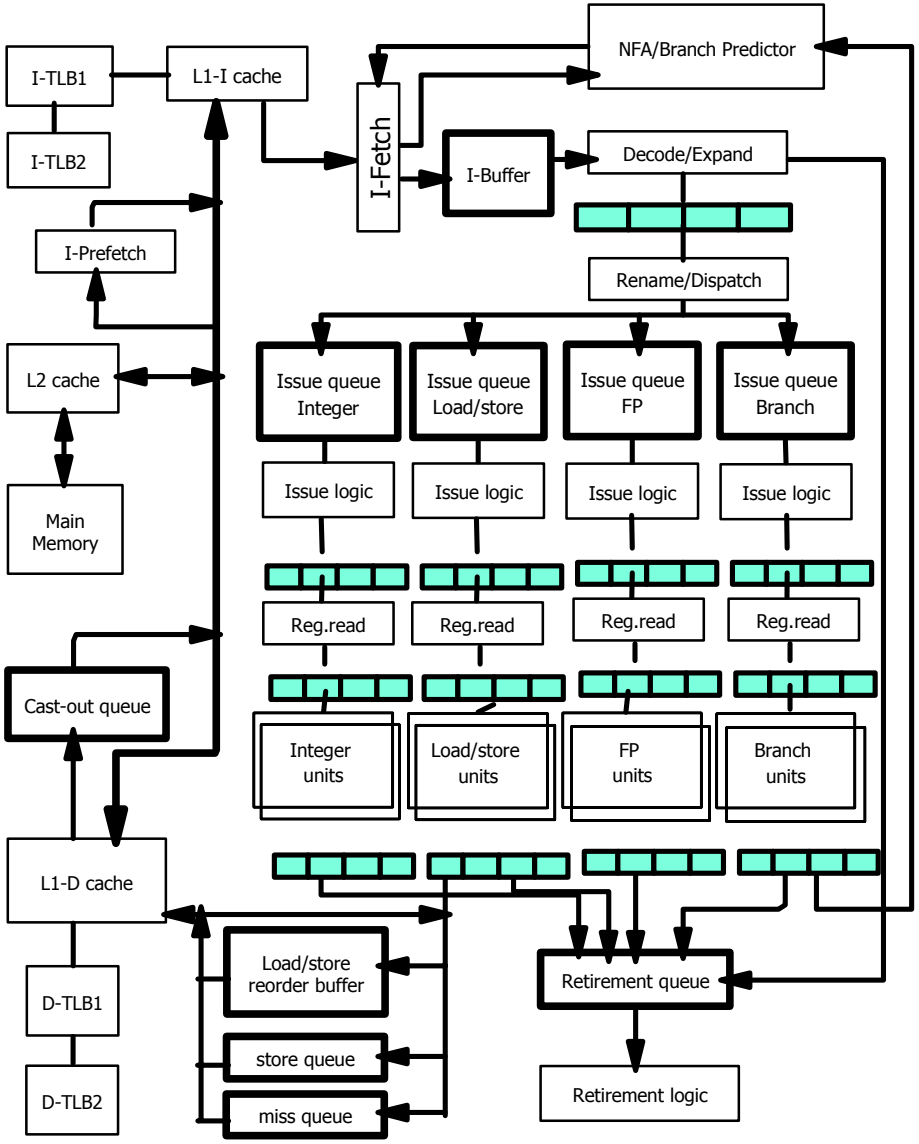
Issue/Execute Stage: MEM



stage_mem_exec.macros/stage_mem1_exec.macros
load/store instruction execution

1. If MEM is already stalled for some reason, check if that is resolved. If so, resume MEM.
2. Calculate #insns executable this cycle, exit if none.
3. For each insn in mem issue queue:
    1. If INORDER_BRANCHES, stall if there is branch ahead.
    2. If operands not ready, exit.
    3. Handle non-mem type insns.
    4. Check bank conflicts
    5. Check DTLB1/DTLB2, stall if miss.
    6. For store, insert into storeq.
    7. For load, first search storeq to see if match any existing stores, if yes, bypass. Otherwise, insert into reorder queue.
    8. Check L1 dcache / trailing edge / L2 dcache. If miss, move insn from memq to dmissq, which is ordered by the time when data is ready.
4. Set results to be available after a number of cycles depending on the instruction latency
5. Remove insn from mem issue queue.

WB/Retire Stage

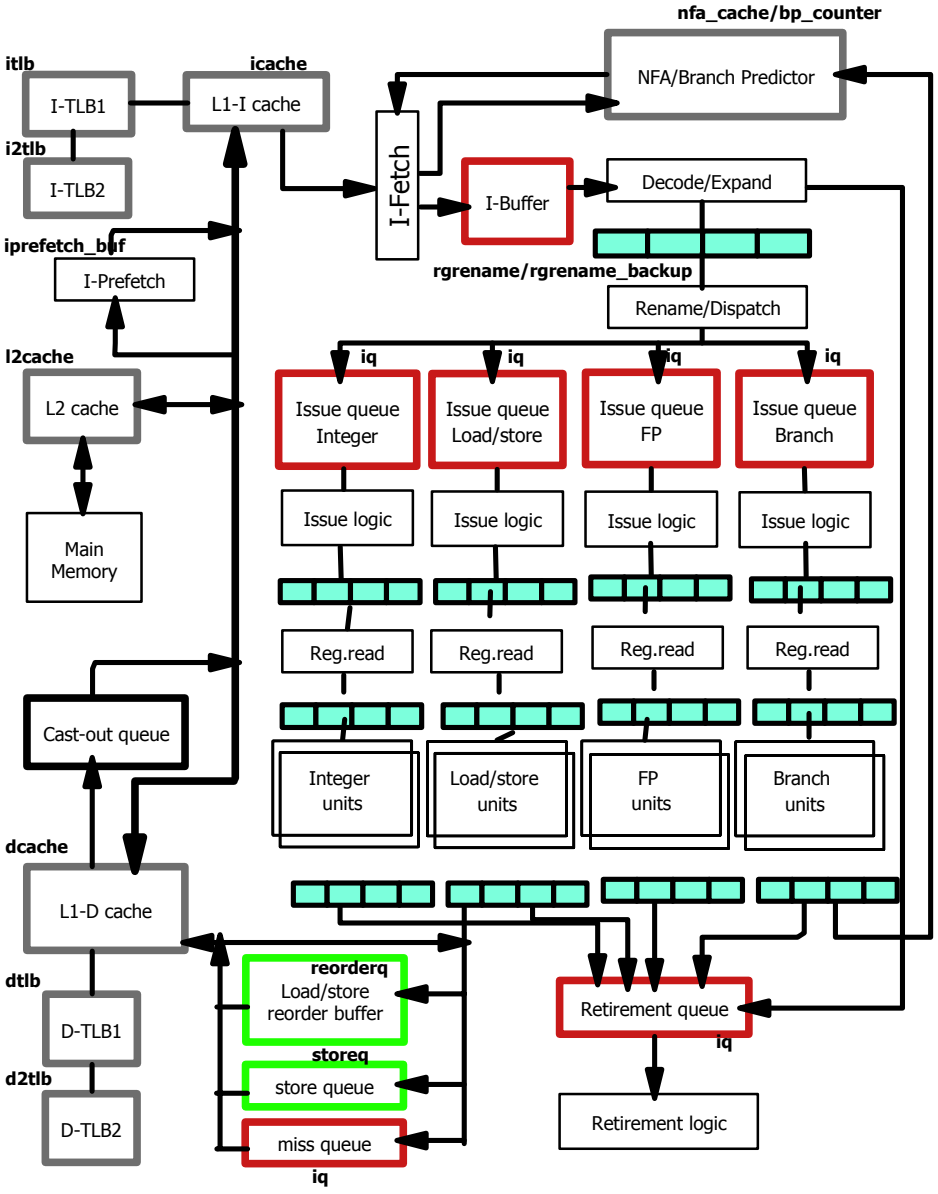


1. If store, remove from storeq
2. If load, remove from reorderq, check if there is a load/store conflict. If yes, flush the pipeline (reset.macros)
3. Update branch history
4. Remove instruction from retireq

stage_retire.macros

retire instructions

Data Structures: Instruction Queue, Array, Reorderq/Storeq



iq.h

Instruction queue

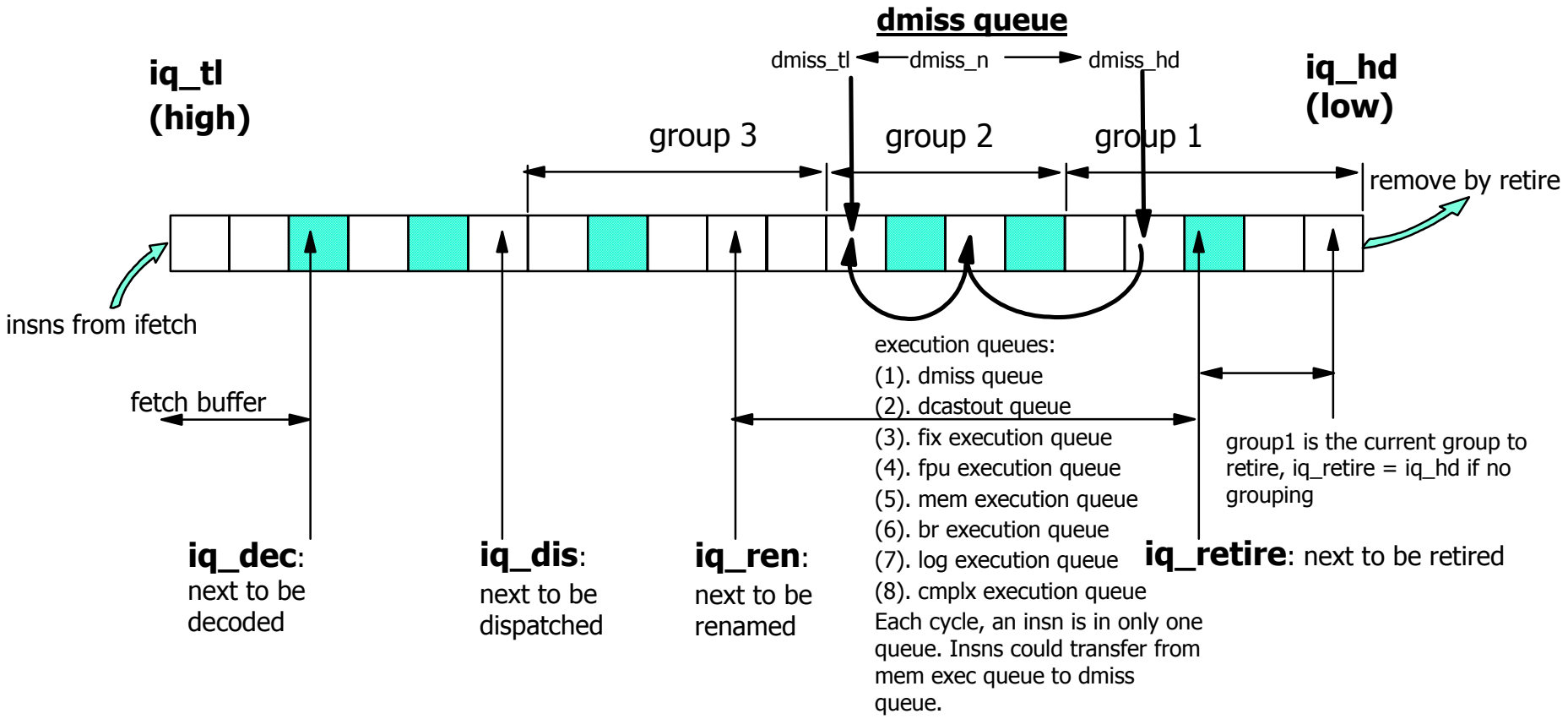
array.c/h

array definition and implementation

block_memq.macros

reorderq/storeq

Data Structures: Instruction Queue



1. All inflight insns are stored in iq throughout its life span (from fetch to retire)
2. New insns are inserted into iq_tl, retired instructions are removed from iq_hd
3. Sub queues (ibuffer, issue queues, retireq, missq, etc.) are formed through:
    1. head pointer, tail pointer, element count (e.g. fpu_hd, fpu_tl, fpu_n)
    2. next pointer

Instruction Queue (cont'd)

```
#if(!defined(TIMELINE_SIZE))
  #define TIMELINE_SIZE      1024
#endif

struct xiq_info {
  xdep_prep_info *  entry;
  unsigned                   iaddr;
  unsigned                   daddr;
  xarch *                    out_regs;
  xcycle                     cycle;
  unsigned                   start;
  unsigned                   early;

  unsigned                   prop0;
  unsigned                   prop1;
  unsigned                   prop2;
  unsigned                   propl;
  unsigned                   iword;

  xiq_regidx                 reg_idx;

  xiq_idx                    next;
  unsigned char              trauma;
  unsigned char              slot;
  unsigned char              cluster;


  #if(USING_FF52_SEGS )
  unsigned                   dseg;
  #endif

  #if( TRACE )
    xstatus                  status;
  #endif
  #if( TRACE || DUMP_DMISS || TIMELINE )
    unsigned                 count;
  #endif
  #if( TIMELINE )
    xcycle          tl_start;
    xcycle          tl_complete;
    char
tl_status[TIMELINE_SIZE];
  #endif
};
```

cycle: time when data/output is ready.
for non load/store instructions:

xiq_cycle = cycle + latency + RF_DELAY

for load,

(1). If data can be forwarded from storeq

xiq_cycle = cycle + latency + RF_DELAY + STOREQ_FWD_DELAY, tmp_fwd + STOREQ_FWD_DELAY

(2). If data found in L1 dcache

xiq_cycle = cycle + L1 latency + RF_DELAY (L1 latency is not defined?)

(3). if data found in L2 cache

xiq_cycle = cycle + L2 latency + RF_DELAY

(4). if data found in memory

xiq_cycle = cycle + mem_latency + RF_DELAY

In situation (3) and (4), xiq_cycle is the earliest time that data could appear on the bus. The final latency will include the bus and queueing latency as well. Note that the bus transactions (dmiss queue) is ordered by xiq_cycle. Each dmiss entry can use the bus only after current cycle passes xiq_cycle.

start: address of the first insn in the current fetch block, used very rarely. in flush_mispredicted.macros, to fix up NFA table, you need the address that was originally used in ifetch to index into the NFA. that address is stored in **start**.

early: used in two situations:

1. Bank conflict:

xiq_early(tiq[tmp_at[thread_exec]]) = cycle + DCACHE_INTERLEAVE_PENALTY

2. Load overlaps a store in the storeq. But data is not ready yet. (GP) Will recirculate after 7 cycles.

xiq_early(tiq[tmp_at[thread_exec]]) = cycle + 7;

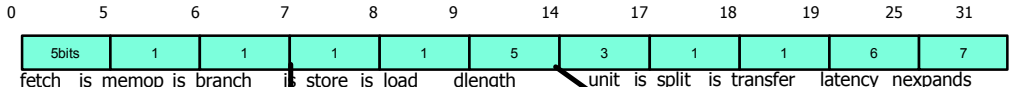
Instruction Queue (Cont'd)

```
#if(!defined(TIMELINE_SIZE))
  #define TIMELINE_SIZE      1024
#endif

struct xiq_info {
  xdep_prep_info *  entry;
  unsigned                  iaddr;
  unsigned                  daddr;
  xarch *           out_regs;
  xcycle                    cycle;
  unsigned                  start;
  unsigned                  early;

  unsigned                  prop0;
  unsigned                  prop1;
  unsigned                  prop2;
  unsigned                  propl;
  unsigned                  iword;

  xiq_regidx                reg_idx;

  xiq_idx           next;
  unsigned char             trauma;
  unsigned char             slot;
  unsigned char             cluster;


  #if(USING_FF52_SEGS )
  unsigned                  dseg;
  #endif

  #if( TRACE )
    xstatus         status;
  #endif
  #if( TRACE || DUMP_DMISS || TIMELINE )
    unsigned                count;
  #endif
  #if( TIMELINE )
    xcycle          tl_start;
    xcycle          tl_complete;
    char
tl_status[TIMELINE_SIZE];
  #endif
};
```

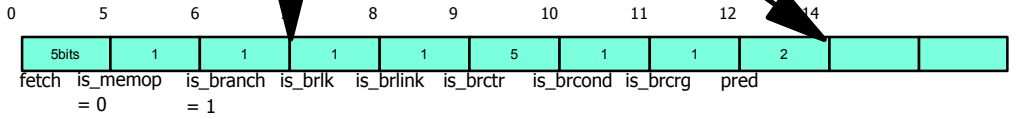
prop0, prop1, prop2 are inherited from predecode stage. propl contains runtime info.

prop0

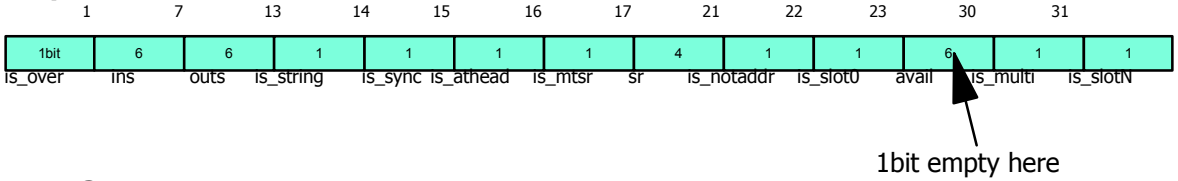
memop:



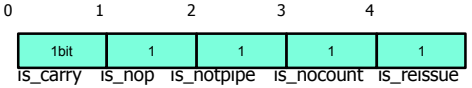
branch:



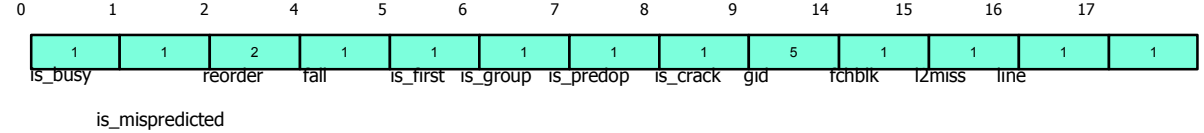
prop1



prop2



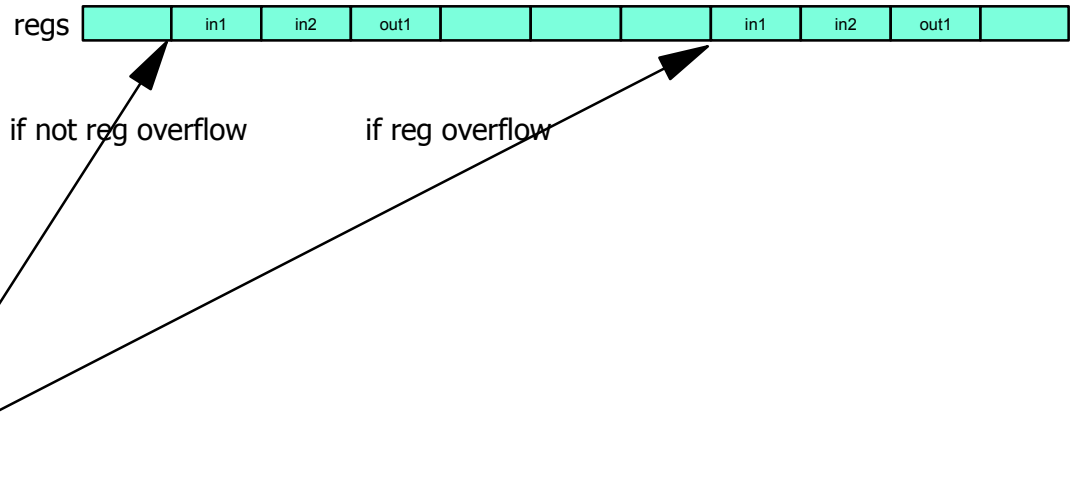
propl



Instruction Queue (Cont'd)

```
#if(!defined(TIMELINE_SIZE))
  #define TIMELINE_SIZE      1024
#endif

struct xiq_info {
  xdep_prep_info *  entry;
  unsigned                  iaddr;
  unsigned                  daddr;
  xarch *           out_regs;
  xcycle                    cycle;
  unsigned                  start;
  unsigned                  early;

  unsigned                  prop0;
  unsigned                  prop1;
  unsigned                  prop2;
  unsigned                  propl;
  unsigned                  iword;

  xiq_regidx                reg_idx;

  xiq_idx                   next;
  unsigned char             trauma;
  unsigned char             slot;
  unsigned char             cluster;


  #if(USING_FF52_SEGS )
  unsigned                  dseg;
  #endif

  #if( TRACE )
    xstatus         status;
  #endif
  #if( TRACE || DUMP_DMISS || TIMELINE )
    unsigned                count;
  #endif
  #if( TIMELINE )
    xcycle          tl_start;
    xcycle          tl_complete;
    char
tl_status[TIMELINE_SIZE];
  #endif
};
```

reg_idx: index to this instruction's operand and output register ids

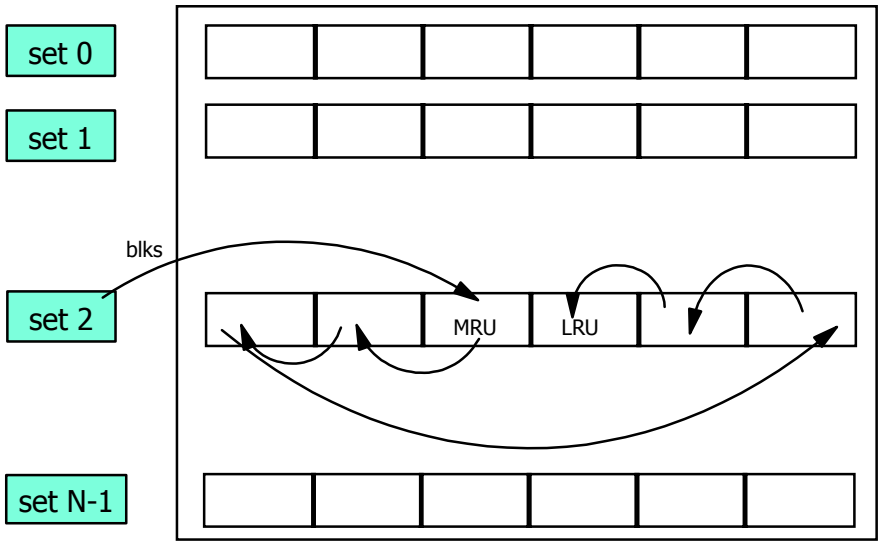


slot: index to the slot in reorderq/storeq, only used by load/store

cluster: not used.

dseg: segment-adjusted daddr, used when USING_FF52_SEGS.

Data Structures: Array



Array structures include:
(1). Data/Instruction caches
dcache, icache, l2cache
(2). TLBs
dtlb, itlb, d2tlb, i2tlb
(2). NFA/BTB table
nfa
(3). Counter table for counter-based branch insns
bp_counter

```
#define ARRAY_LOOKUP        0
#define ARRAY_HIT_UPDATE    1
#define ARRAY_MISS_UPDATE   2

#define ARRAY_READ          4
#define ARRAY_WRITE         8
```

Each array access specifies two aspects of operations:
1. Read/Write: this decides whether the data should be marked dirty.
2. Whether the tag/LRU stack should be updated
   (1). ARRAY_LOOKUP: probe only, no update to tag or LRU stack
   (2). ARRAY_HIT_UPDATE: when hit, update the LRU stack
   (3). ARRAY_MISS_UPDATE: when miss, update tag and LRU stack

array_access returns pointer to the cache line to the caller for further handling (content update, etc.)

Data Structures: Reorderq/Storeq

```
/* storeq */
  unsigned   storeq_hi = 0;
  unsigned   storeq_retired = 0;
  unsigned   storeq_lo = 0;
  unsigned   storeq[STOREQ_SIZE][5];
   /* storeq[][0] is address of first byte touched */
   /* storeq[][1] is address of last byte touched + 1 */
   /* storeq[][2] is the queue id of the store */
   /* storeq[][3] is the time at which the data is transferred
*/
   /* storeq[][4] is the bit vector of reorderq entries waiting
for
   this store to complete, it is also used to store (after
retirement)
   the time of retirement */


/* reorderq */
  unsigned   reorderq_left = REORDERQ_HIWATER;
  unsigned   reorderq_hi = 0;
  unsigned   reorderq_lo = 0;
  unsigned   reorderq[REORDERQ_SIZE][4];
  unsigned   stats_reorderq[ REORDERQ_LENGTH + 1];
  unsigned   stats_reorderq_stall = 0;
  unsigned   stats_reorderq_conflict = 0;
  unsigned   stats_reorderq_total = 0;
   /* reorderq[][0] is the address of the first byte touched
*/
   /* reorderq[][1] is address of last byte touched + 1 */
   /* storeq[][2] is the queue id of the store */
   /* storeq[][3] is the time at which the data is transferred
*/
```

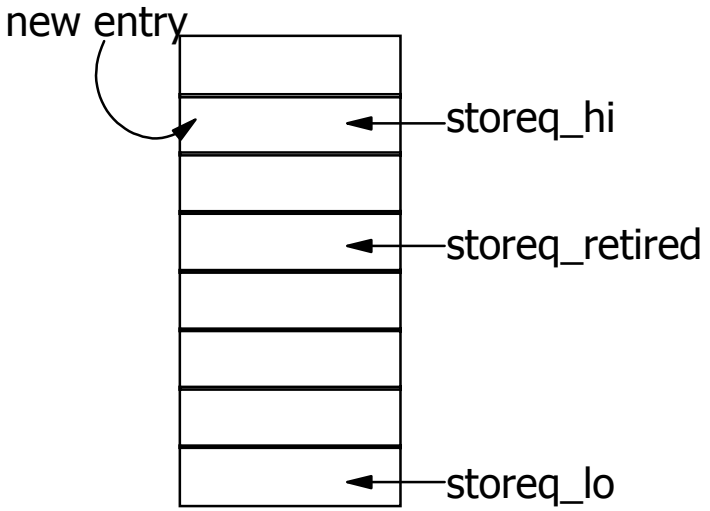
Functionality:

When a load is executed its address is put in the reorderq.
When a store is executed its address and time of data arrival are put in the storeq. These structrures are used for various purposes:

- if a load address collides with a store address already inte storeq, then the model assumes that the load gets its value from the storeq entry and behaves appropriately
- if there is a load/store address collision, but the store data has not yet become available, then the load will be delayed until some time after the store data arrives
- if a store is executed, and its address collides with some previously executed load, which succeeded it in the instruction stream (i.e. the load should have read the value of the store, but was incorrectly data-speculated above the store) then load was inccorectly executed, and some appropriate flushing action is taken.

Implementation:

storeq and reorderq are implemented as arrays. so when an entry is freed, the queue/array needs to be compacted. This greatly complicates the reorderq/storeq implementation.



Data Structures: Register

physical register file, **phys[]**:

ARCH_TOTAL = CLASS_MAX * ARCH_MAX
PHYS_TOTAL = CLASS_MAX * PHYS_MAX

```
typedef unsigned    xcycle;
xcycle              phys[PHYS_TOTAL];
#if( GETCPI )
unsigned char               src_phys[PHYS_TOTAL];
#endif
```

The physical register file keeps track of the cycle when a value in a physical register becomes available to dependent instructions. When an output architected register is renamed, and a physical register is allocated to it, that physical register is initially set to UINT_MAX, After the instruction executes, the physical register is set to the current cycle plus some delay based on the avail-distance (approx. latency) of the instruction.

register rename map, **rgrename[]**:

```
typedef unsigned short      xphys_reg;
xphys_reg                   rgrename_backup[ARCH_TOTAL];
xphys_reg                   rgrename[ARCH_TOTAL];
```

This contains the architected to physical mapping that is valid for instructions being renamed. It is recommended that you look at "Register renaming and dynamic speculation: an alternative approach", M. Moudgill, K. Pingali and S. Vassiliadis, MICRO 26, 1993

backup map **rgrename_backup[]**:

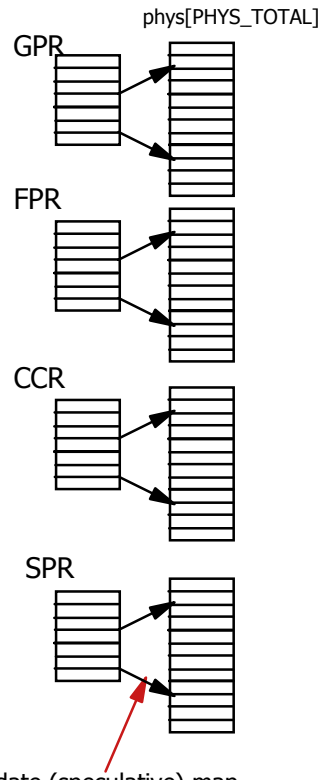
```
xphys_reg                   rgrename_backup[ARCH_TOTAL];
```

When a non-taken path starts being modelled, the current rgrename map is copied to the rgrename_backup. When the misprediction is resolved, the rgrename map is restored from this copy.
- to restore the rgrename map after a mispredict

inorder map, **inorder[]**

```
xphys_reg                   inorder[ARCH_TOTAL];
```

This contains the inorder map, i.e. the architected to physical rename map that was used by the next instruction to retire on a taken path. It is used to figure out which physical registers can be freed. It is also used to compute the rgrename map if arbitrary instructions are flushed

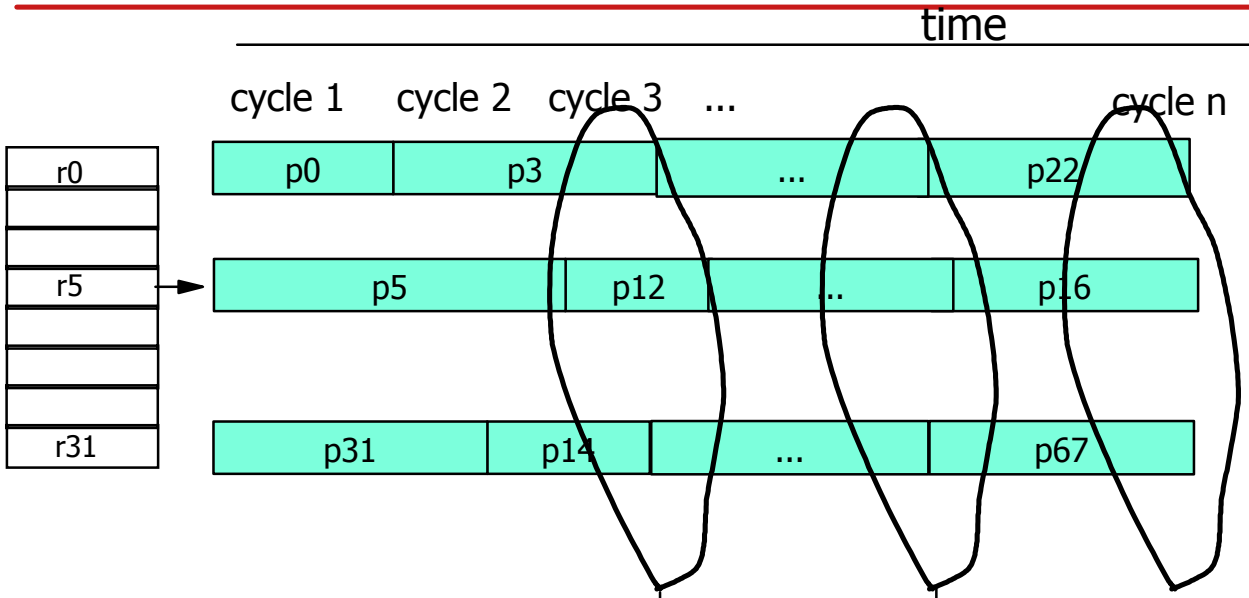


rgrename[ARCH_TOTAL] : normal up-to-date (speculative) map

rgrename_backup[ARCH_TOTAL]: backup map for misprediction handling

inorder[ARCH_TOTAL]: backup map for retire handling

Register Renaming Explained



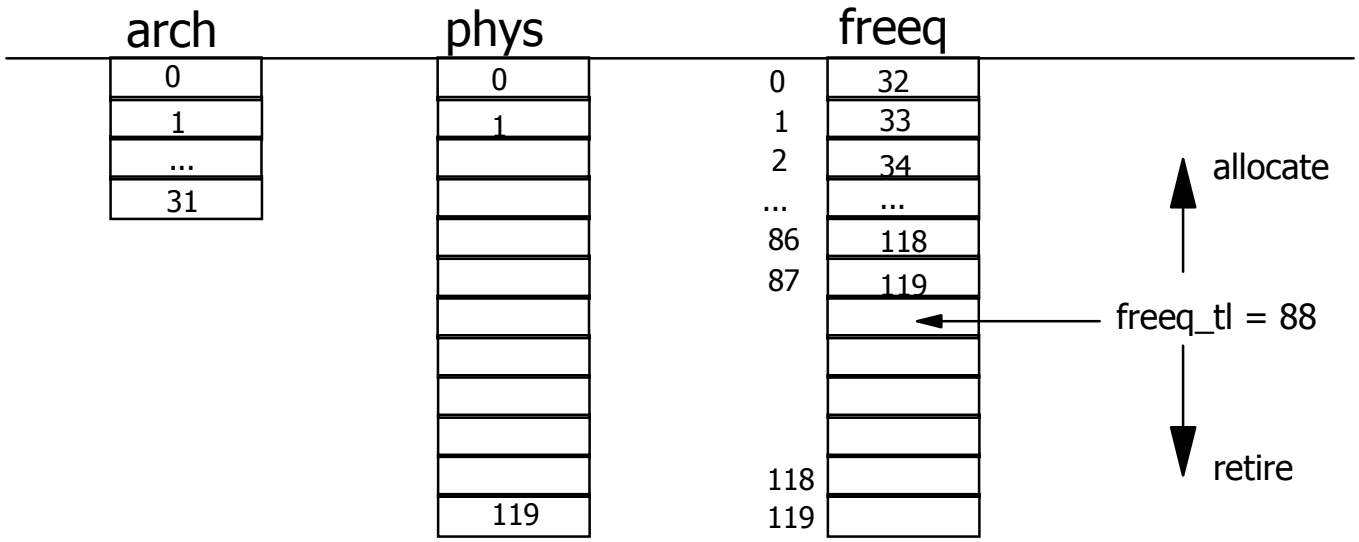
rgrename:
up-to-date speculative map
updated when a new preg is allocated

rgrename_backup:
map prior to a mispredicted branch

inorder:
up-to-date non-speculaitve map
updated when a preg is deallocated

1. Think about tags in cvs version control system!
2. When a new physical register is allocated, update rgrename
3. When a physical register is deallocated, update inorder

Registers and Free List



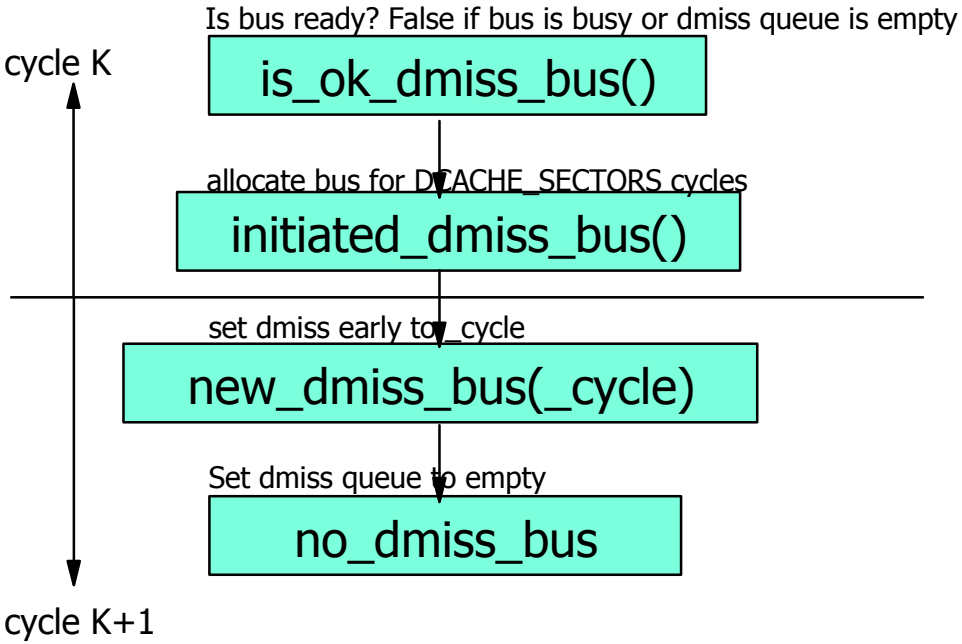
1. Initially, architectural register 0 - 31 are mapped to physical register 0 - 31, and physical registers 32 - 119 are free (as shown above)
2. Pointer freeq_tl points to the tail of free physical registers
3. When a preg is allocated, freeq_tl move up one slot
4. When a preg is deallocated, freeq_tl move down one slot
5. Freeq_tl operates similar to a stack

Bus Model

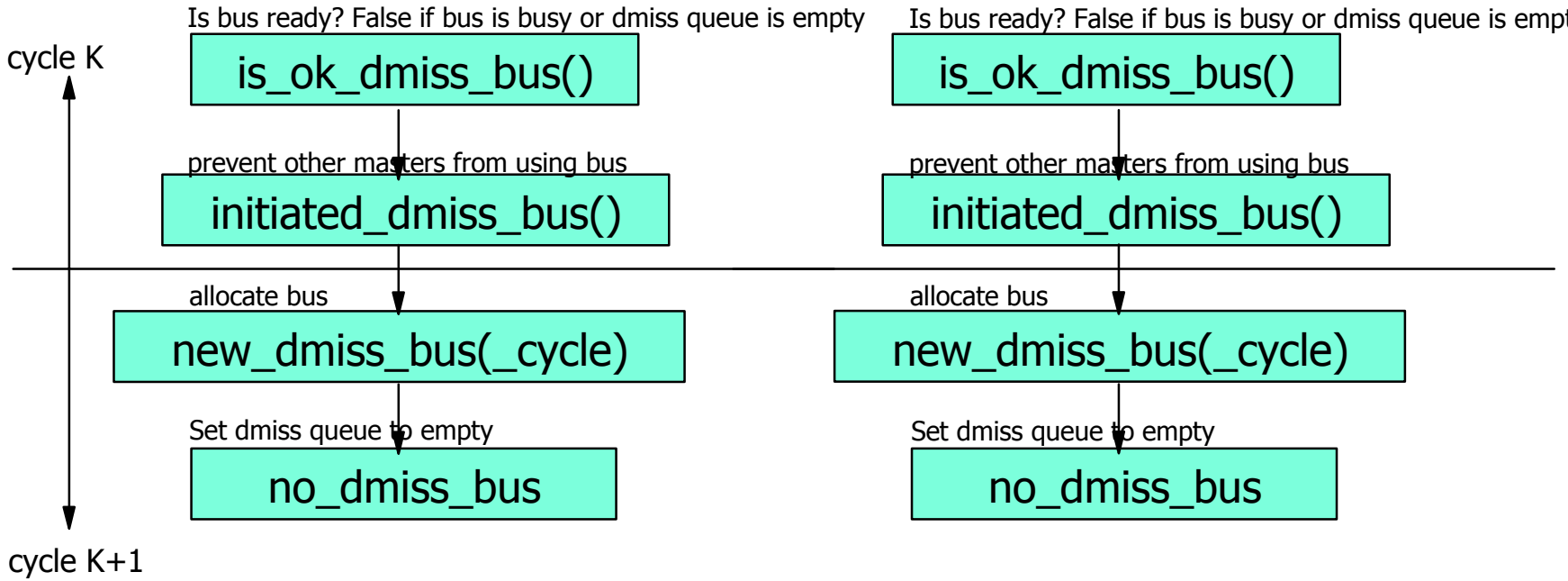
- Currently only L1/L2 bus is modeled
- L1/L2 bus is shared by imiss, dmiss, and dcastout
- Dmissq is ordered according to the time when data is ready
- For dmiss, when data is ready
  - Check if bus is available by is_ok_dmiss_bus()
  - Allocate bus for DCACHE_SECTORS cycles to transfer all sectors by initiated_dmiss_bus(), no other transactions are allowed during this period
- Dcastout and imiss are handled similarly
- More detailed bus / DRAM model is underway

Bus Model (out-dated)

```
/* The activities that access the bus are imiss, [iprefetch,]        castout, and dmiss
/* The earliest such an activity can initiate/complete is controlled by
/* dcastout_early, dmiss_early, and ifetch_miss_till
/*
/* The amount of time an activity occupies the bus is given by    */
/* DCASTOUT_OVERHEAD, DCACHE_SECTORS, 1
*/
/*
*/
/* The next request for a dcastout is:                                */
/*          - UINT_MAX if empty,                                      */
/*          - ASAP else
*/
/*
*/
/* The next request for a dmiss is:                                   */
/*          - UINT_MAX if empty,                                      */
/*          - xiq_cycle( iq[dmiss_hd] )                               */
/*
*/
/* The next request for a imiss is:                                   */
/*          - UINT_MAX if none in progress                            */
/*          - ifetch_miss_till else                                   */
/*
*/
/*********************************************************************/
#define is_ok_dmiss_bus()                   dmiss_early <= cycle )

#define initiated_dmiss_bus()                                          \
 {                                                                                 \
   dcastout_early = cycle + DCASTOUT_OVERHEAD;                         \
   ifetch_miss_till = max(ifetch_miss_till, dcastout_early);           \
 }

#define new_dmiss_bus(cycle_) dmiss_early = (cycle_);
#define no_dmiss_bus()                      dmiss_early = UINT_MAX;
#define is_ok_dcastout_bus()    ( dcastout_early <= cycle )

#define initiated_dcastout_bus()                                       \
 {                                                                                 \
   /* no dmiss in progress */                                          \
   assert( dmiss_loaded == DCACHE_SECTORS);                            \
   dcastout_early = cycle + DCASTOUT_OVERHEAD;                         \
   dmiss_early = max( dcastout_early, dmiss_early);                    \
   ifetch_miss_till = max( ifetch_miss_till, dcastout_early);          \
 }

#define no_dcastout_bus()                   dcastout_early = UINT_MAX;
#define initiated_ifetch_bus()                                         \
 {                                                                                 \
   dcastout_early = cycle + DCASTOUT_OVERHEAD;                         \
```

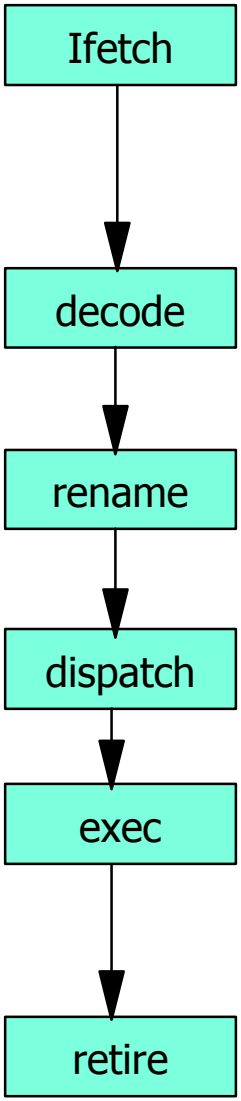


Bus Model in SMT Mode (out-dated)



1. Initiated_dmiss_bus and new_dmiss_bus need to be atomic
2. stats_il2miss[thread_ifetch]++;
3. ifetch_stalled_till[thread_ifetch] = IFETCH_STALL_CACHE;
4. new_ifetch_bus(thread_ifetch, max(cycle, dtlb_miss_till) + IMEM_LATENCY);
5. Rename
   - I. Check if enough rename registers available, if not, stall until available.
   - II. Rename architectural registers to physical registers.
   - III. If the instruction is a mispredicted branch instruction, check if all operands are ready. If yes, resolve the branch and start fetching from the right path from next cycle.
   - IV. Note: registers in different class are renamed seperately.
6. Dispatch
   1. Place renamed IOPs into the corresponding issue queue. If a given operation can not be placed in the issue queue (i.e. the queue is full), stall the stage until available.

Branch Handling



if ifetch_in_mispredicted is true, and ifetch_mispredict_done is true, then
1. set ifetch_in_mispredicted = FALSE, revert trace reader to taken
2. if ifetch_mispredict_done_early is done early, then stall 1 cycle, otherwise, stall 1 + MISPREDICT_RECOVERY_CYCLES cycles.
3. flush_mispredicted.macros

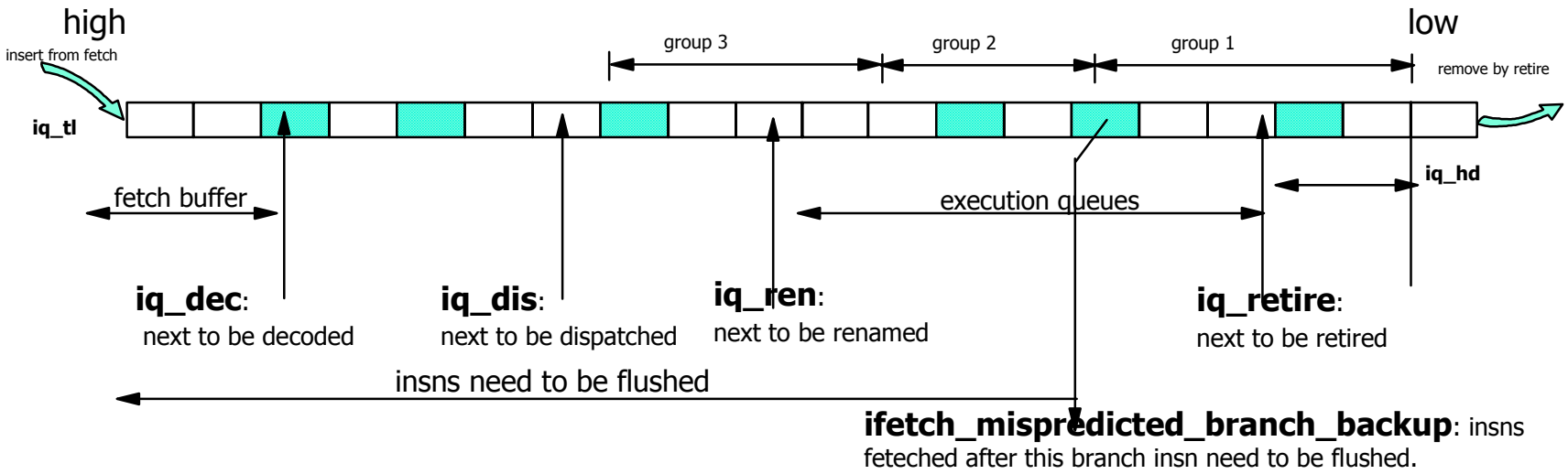
fetch, then look up branch predictor. If mispredicted, then
1. ifetch_in_mispredicted = TRUE, ifetch_mispredict_done = FALSE, ifetch_mispredict_done_early = FALSE, ifetch_mispredicted_branch = tmp_at, ifetch_mispredicted_branch_backup = tmp_at, branch_pred_delta = 0
2. reader swap to not taken
3. push return address to RAS if it's a link branch
4. update NFA, exit fetching

If EVALUATE_MISPREDICTED_RENAME, then check if all operands of the branch are ready. If yes, set ifetch_mispredicted_branch = IQ_NULL, ifetch_mispredict_done = TRUE, ifetch_mispredict_done_early = TRUE.

If all operands are ready,
1. update stats such as bp_exec_total, bp_exec_pred, bp_exec_miss, bp_exec_bc_total, bp_exec_bclr_total, bp_exec_bcctr_total, bp_exec_bc_miss, bp_exec_bclr_miss, bp_exec_bcctr_miss, branch_pred, branch_pred_delta,
2. update counter table for counter-based branch insns
3. if this is the first mispredicted branch, update ifetch_mispredicted_branch = IQ_NULL, ifetch_mispredict_done = 1
4. set the time of ready for output regs
5. remove the branch from br queue.

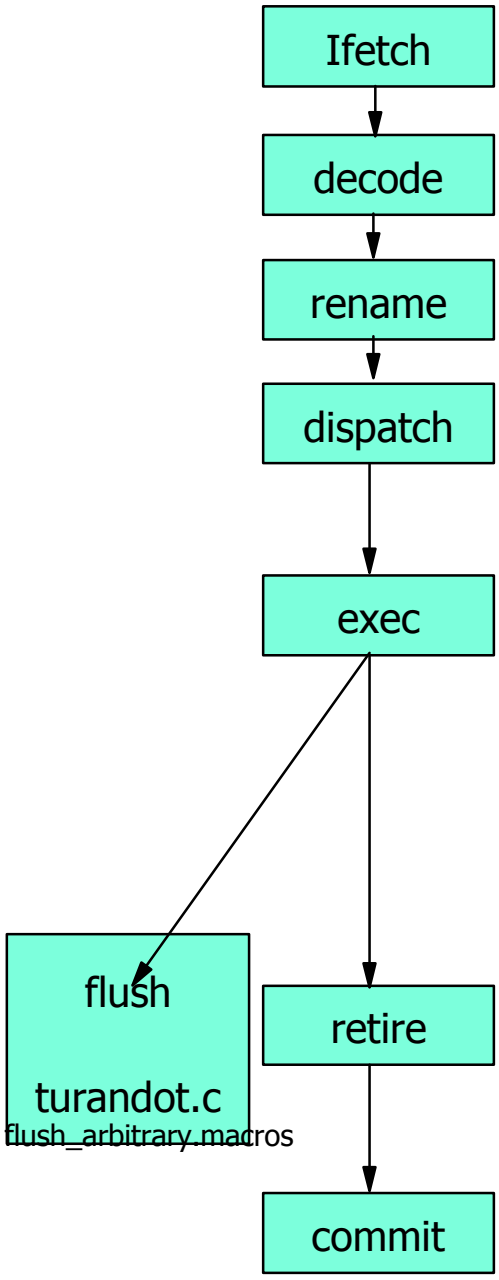
update branch history in the branch predictor

Misprediction Flush (flush_mispredicted.macros)



1. If do_flush and xiq_is_mispredicted(iq_flush) are both true, reset do_flush because in this situation flush_arbitrary is not necessary and are covered by flush_mispredicted.
2. Reset branch_pred, branch_pred_delta, ibuf_left, fchblk_left
3. Check to make sure that all insns unflushed (from ifetch_mispredicted_branch_backup to iq_hd) are not speculative
4. Check to make sure that all to-be-flushed insns (from ifetch_mispredicted_branch_backup + 1 to iq_tl) are speculative
5. Reset ifetch_branch_history to ifetch_mispredicted_history_backup
6. Reset iq_tl, iq_fetch, iq_dec, iq_dec_1, iq_dec2, iq_dec3, iq_dec4
7. Fix the NFA table for branch insn at ifetch_mispredicted_branch_backup. If branch is taken, write the target address into NFA entry, otherwise, reset the corresponding NFA entry to 0
8. If this mispredicted branch is resolved early in rename or dispatch stage, then skip the following steps
9. Reset retireq_left, groups_left, decode_reissue_groups, groups_gid
10. Walk through unflushed insns to adjust groups_left
11. Reset iq_ren, iq_dis
12. Reset rgrename to rgrename_backup
13. Walk through flushed insns that have been renamed, reclaim the physical regs used by them
14. Drain storeq (why?)
15. Flush speculative entries in storeq and reorderq
16. If fix execq is not empty, remove entries corresponding to to-be-flushed (speculative) fix insns
17. Do 10 for fix1, fpu, fpu1, br, cmplx, log, mem, mem1, dmiss queue.
18. If dmiss_thread == thread_flush, reset dmiss_address, dmiss_is_store, dmiss_loaded

Load/Store Handling



Nothing special for memory insns in fetch stage

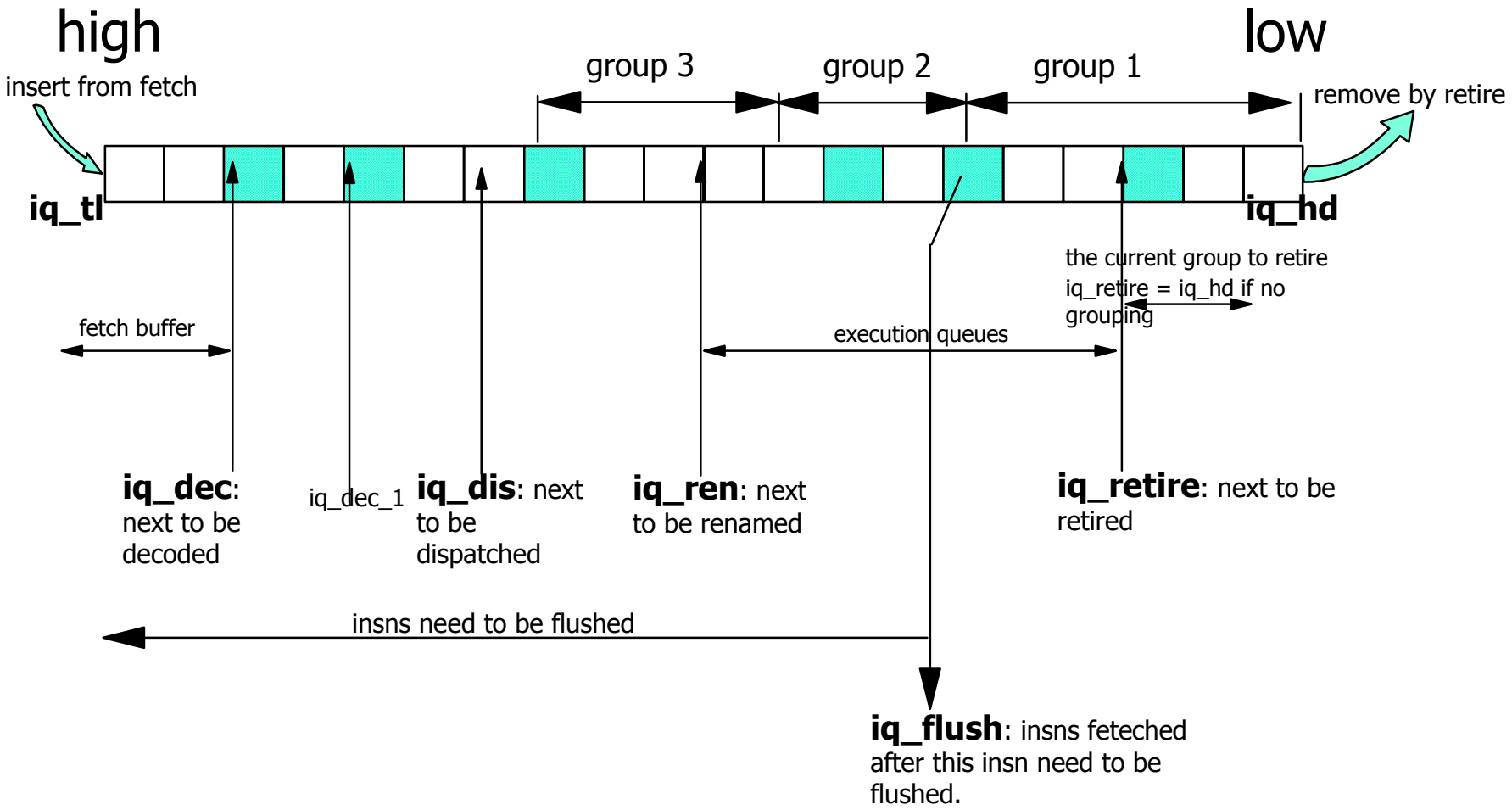
If STOREQ_AT_DISPATCH, allocate a storeq entry for each store using get_slot_storeq
If REORDERQ_AT_DISPATCH, allocate a reorderq entry for each load using get_slot_reorderq
if STOREQ_AT_DISPATCH, and current insn is a transfer op (each store is splitted into a store op and a transfer op), then use the same storeq entry as the store op. (use_prev_slot_storeq)

1. Check if space available in dmiss queue and castout queue, if not, stall memory pipe.
2. Issue ready instructions in the memory issue queue in oldest-first order. For each memory instruction:
   a. Check DTLB1/DTLB2, stall if miss
   b. Place store into store queue.
   c. For load, first search storeq to see if match any existing stores, if fully hit, bypass. If partially hit, (1). if load/store are not in the same group, recirculate the load (2). if load/store are in the same group, then flush the pipeline.
   d. place into reorder queue according to the earliest time the data can return.
   e. Check Dcache1, if miss, check trailing edges, then Dcache2, stall if any miss
   f. If TLB miss, the whole memory pipeline stalls. Otherwise, the data latency is calculated depending on whether L1/L2 hit/miss etc.

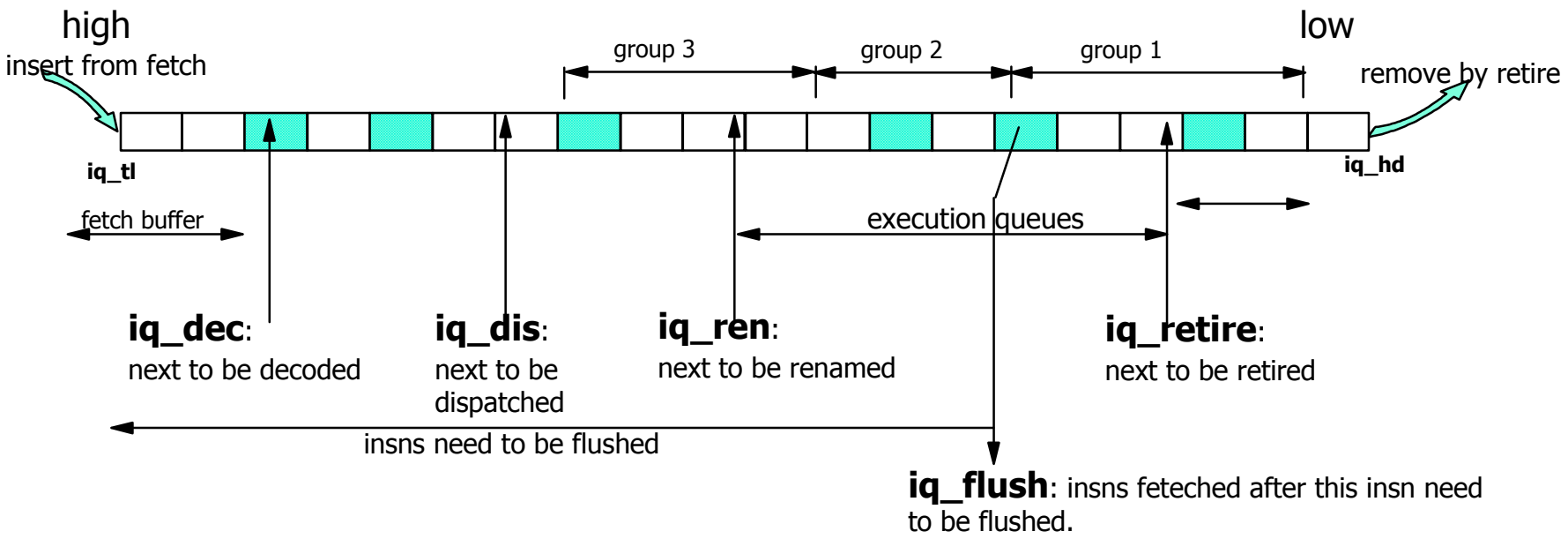
For a store, if data is ready, and !COMMIT_STORES_LATE, then write data to cache, and reclaim the storeq entry (pop_storeq). If COMMIT_STORES_LATE, then just retire the storeq entry (retire_storeq).
For a load, first reclaim the reorderq entry (pop_reorderq). If there is a reorder conflict, (xiq_reorder(iq[iq_retire]) == 2), then reset the pipeline (reset.macros).

if COMMIT_STORES_LATE, write data to cache, and reclaim the storeq entry (pop_storeq)

Memory Conflict Flush (flush_arbitrary.macros)

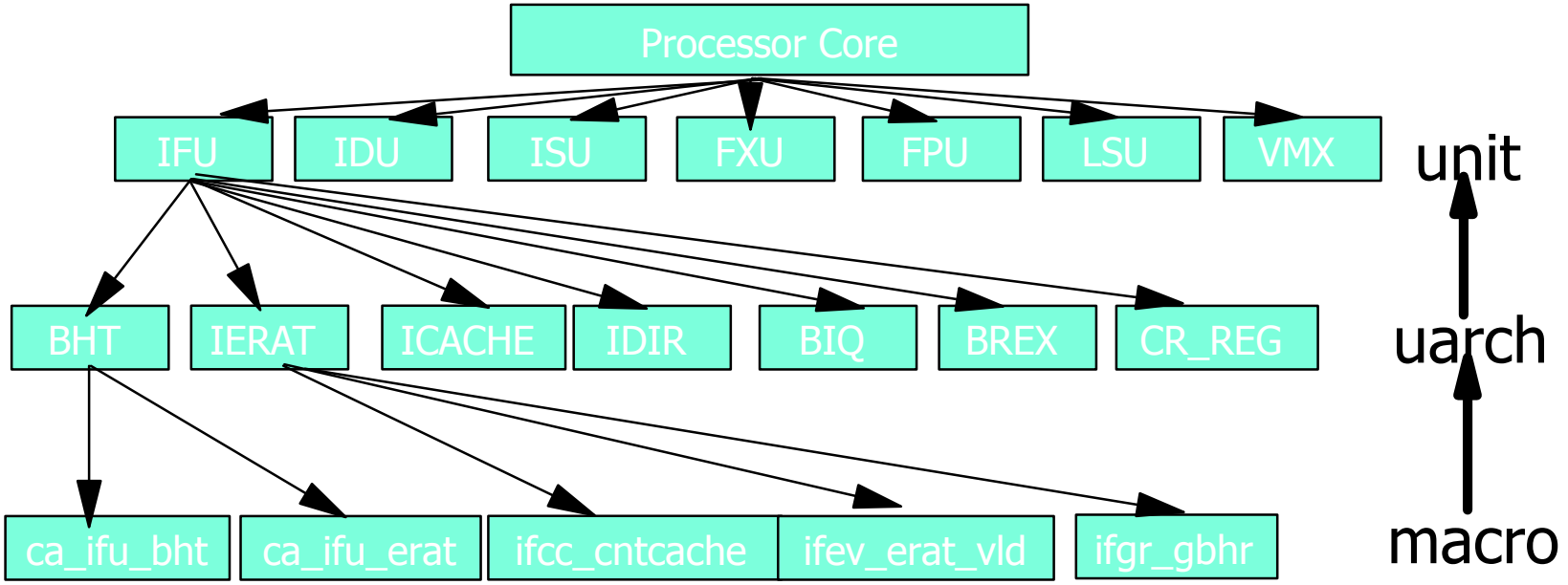


Memory Conflict Flush (flush_arbitrary.macros)

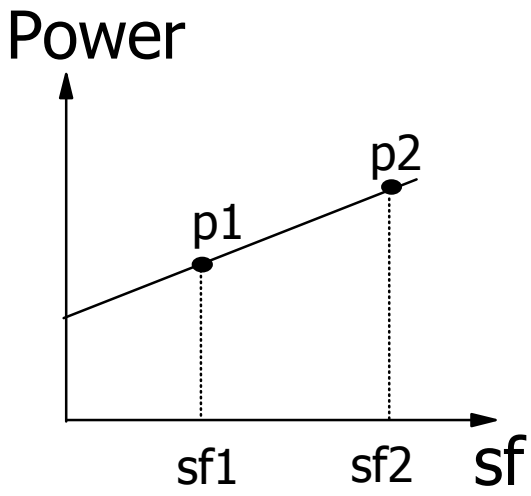


1. Reset groups_left, decode_reissue_groups, groups_gid, branch_pred, branch_pred_delta
2. Reset rgrename to inorder
3. Walk through unflushed insns (from iq_retire to iq_flush), adjust groups_left, branch_pred, branch_pred_delta and ifetch_mispredicted_history_backup, rgrename
4. Walk through to-be-flushed insns (from iq_flush to iq_tl), fix RAS stack
5. Walk through to-be-flushed insns, count #non-speculative insns, reset to taken path if currently fetch is in mispredicted status.
6. Rollback trace reader by #non-speculative insns
7. Reset iq_tl, iq_ifetch, iq_dec, ..., iq_ren, iq_dis
8. Walk through to-be-flushed insns that are renamed (from iq_flush to iq_ren), reclaim physical regs used by them
9. Flush reorderq and storeq entried corresponding to to-be-flushed memory insns
10. If fix execq is not empty, remove entries corresponding to to-be-flushed fix insns
11. Do 10 for fix1, fpu, fpu1, br, cmplx, log, mem, mem1, dmiss queue.
12. If dmiss_thread == thread_flush, reset dmiss_address, dmiss_is_store, dmiss_loaded
13.

Power Model

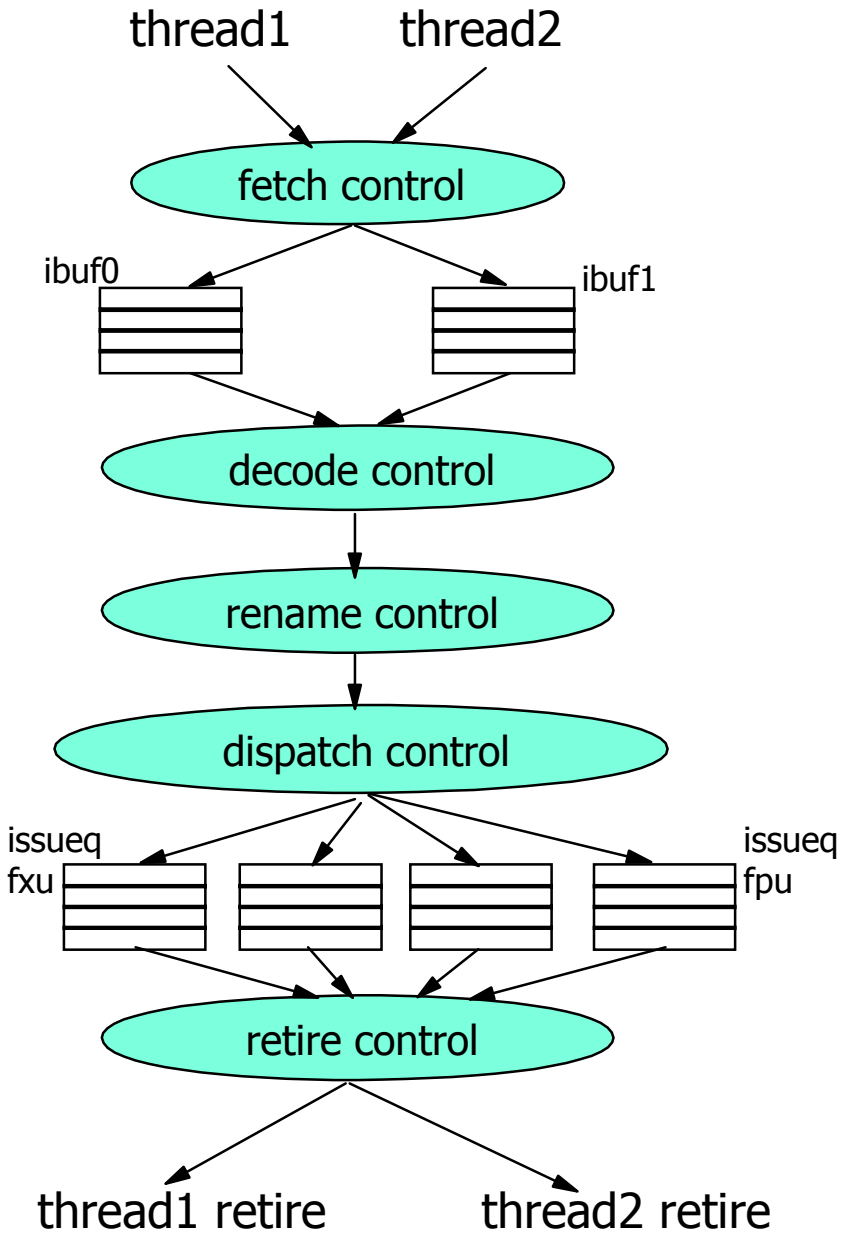


1. Processor core is hierarchically divided into units, uarchs, and eventually macros
2. For each macro, CPAM provides (sf1, p1) and (sf2, p2). Power at other switching factors can be linearly extrapolated
3. Switching factors are extracted from Turandot



init.power.macros
    read in macro power information
power_defs.macros
    some data structures (unit, uarch, macro) and some utility functions
power.macros
    switching factor calculation based on Turandot statistics, and power calculation

SMT Mode



1. Functional units (fxu, fpu, etc) are shared among all threads
2. Architectural registers are duplicated for each thread
3. Other resources (cache, queue, bpred, etc) can be either duplicated, or shared, depending on design choice.
4. At each control point, either one thread can be selected to proceed, or insns from different threads can be selected
   1. Can be as simple as round-robin
   2. Or better policies?

Summary of New Features Added to Original Turandot

- SMT support
- New, simplified array model
- Standalone predecode stages removed
- Ported to linux/x86 and cygwin
- Gzip/bzip/uncompressed trace formats all supported
  - gpffturandot -t sample1.fF sample2.fF
  - gpffturandot -t art.bz2 ammp.bz2
- Cycle-by-cycle power model added in addition to original postprocess model
- Voltage model based on RLC power supply network model added
- Temperature/reliability model

Output Stats from Turandot

- Dump of compile-time parameter definitions
  - @@ DCACHE_ASSOC = 2
  - @@ DCACHE_SECTORS = 2
- Summary stats
  - totals: cycle=121229 insns=100734 memops=44870 retired=100000
- Histogram data
  - @@ ibuf@@ 0:77479 63.91
  - @@ 1:1898 1.56
  - @@ 2:3567 2.94
- Trauma data
  - @@ if_nfa : 4 0 0 0 0 0 0 0 0 0 4
  - @@ if_tlb1: 0 0 0 0 0 0 0 0 0 0 0
  - @@ if_tlb2: 51 0 0 0 0 0 0 0 0 0 51
- Summary progress result reported every MONITOR_CYCLES
- Detailed status reported every cycle
- Timeline / pipeline graph
  - 000000010 [.....F..DE.d.....f.....c.....i2.h...] 000000382 10000170 300162f0 lwz r10,40(r2)
  - 000000011 [.....F..DE.d.....c.....i1.f] 000000382 10000174 00000000 addi r9,r0,0
- Power data
  - unconstrained
  - average power

Todo List

- Functionality enhancement
  - CMP support: preferably allow SMT + CMP together
  - Interpretation-based execution-driven mode
    - Aria instruments code and executes locally, so not portable to non-PPC platforms
    - Interpretation based execution is not restricted to any platform
  - Better bus / DRAM model for bandwidth studies
  - Port to windows and other platforms
- Calibration of Power5 model
  - Performance model validation
    - SMT Turandot vs. Power5
  - Power model validation
    - SMT PowerTimer vs. Power5
    - Temperature model vs. Power5 on-chip temperature sensors
- Modularity/readability enhancement
  - Command line options to replace compile-time options
  - More readable reoderq / storeq models (block_memq.macros)
  - Use of bit fields instead of explicit bit handling in props (iq.h)
  - Removal of the gotos
  - Trauma stats check up
  - ...

Collaboration Opportunities

- How to obtain Turandot/PowerTimer
  - Academic research groups: send a formal email request to pbose@us.ibm.com
- Different collaboration levels
  - Turandot/PowerTimer academia users
    - Stable release version 1.0 is available
  - Turandot/PowerTimer academia co-developers
    - Will be involved in development/validation process
- IBM supports collaboration with academia
  - IBM fellowship awards for graduate students
  - IBM faculty partnership awards (FPA)
  - Sabbatical opportunities
  - summer internship
  - Postdoc position
- Contact
  - Pradip Bose (pbose@us.ibm.com)
  - Zhigang Hu (zhigangh@us.ibm.com)
  - Victor Zyuban (zyuban@us.ibm.com)

Installation

- To install, cd to $ROOT/Sources directory:
  - Modify makefile.defs to match your platform and directory settings
  - Run "bash make_all.bash" to compile the whole turandot package. After completion, an executable named "gpffturandot" will appear in $ROOT/Sources/turandot/src.
- To run the simulator, cd to $ROOT/Sources/turandot/src directory,
  - Run "gpffturandot -t $TRACE", $TRACE can have extensions of "fF", "bz2", or "gz".
  - To change configurations, modify Makefile, and rebuild the simulator.
- In $ROOT/doc directory,
  - TurandotUserGuide.pdf
    - Explanation of compile-time configuration parameters
    - Explanation of summary outputs
    - Explanation of cycle-by-cycle detailed outputs
    - Explanation of timeline / pipeline graph
  - power4.pdf
    - A tech report that details the architecture of power4
  - Turandot_Overview.pdf
    - This document

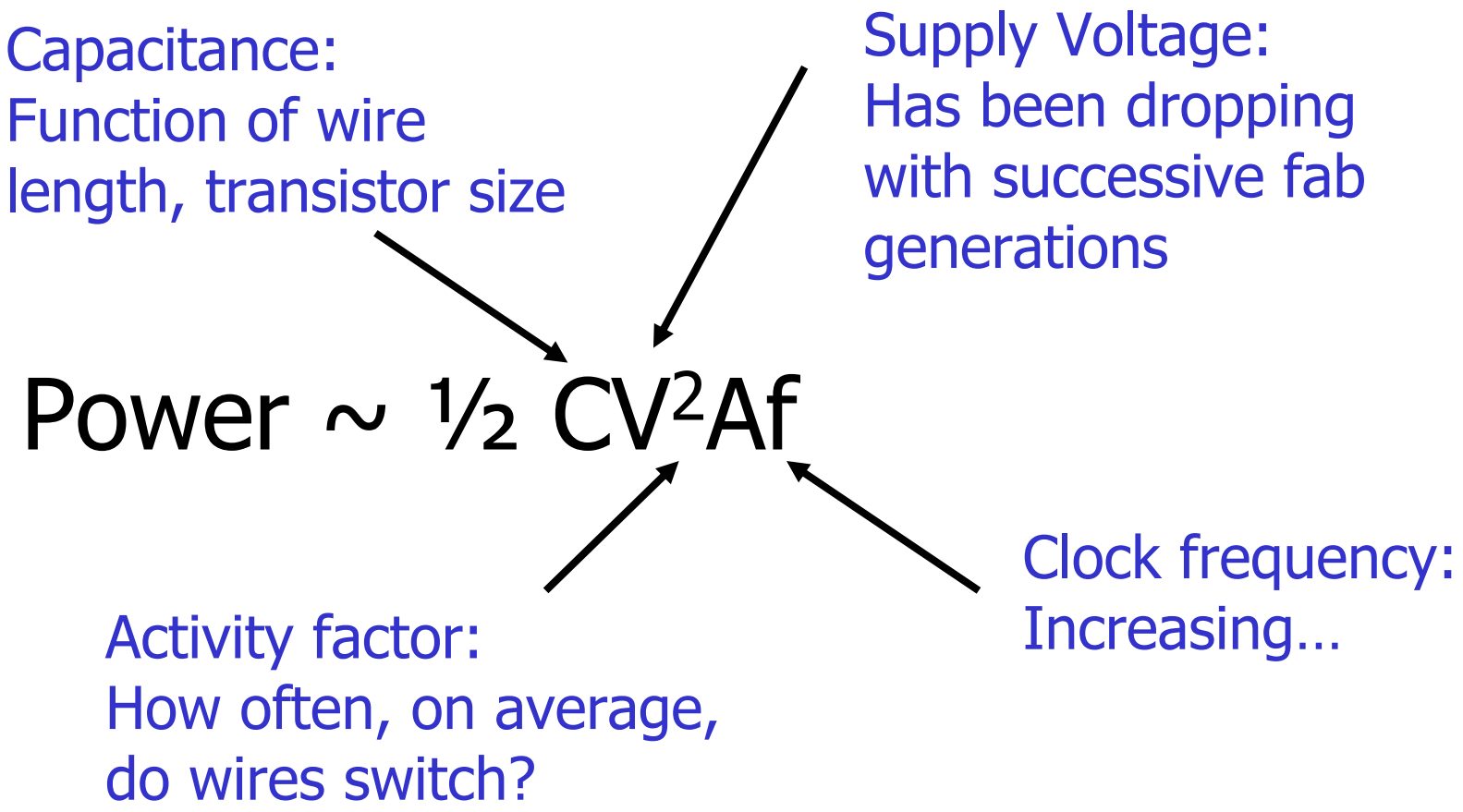
Bibliography: Turandot

- Cathy May, et al., "The PowerPC Architecture: A specification for a new family of RISC processors", second edition, Morgan Kaufmann Publishers.
- J.M. Tendler, J.S. Dodson, J.S. Fields, Jr., H.Le, B. Sinharoy, "Power4 System Architecture", IBM J. RES. & DEV., January 2002, available at http://www.research.ibm.com/journal/rd/461/tendler.pdf
- M. Moudgill, J-D. Wellman, J. Moreno, "Environment for PowerPC Microarchitecture Exploration," IEEE MICRO, May/June 1999, pp. 15-25.
- M. Moudgill, P. Bose, J. Moreno, "Validation of Turandot, a Fast Processor Model for Microarchitecture Exploration,"Proc. IEEE Int'l Performance, Computing and Communications Conference, February 1999, pp.452-457.
- M. Moudgill, J-D. Wellman, J. Moreno, "An Approach for Quantifying the Impact of not Simulating Mispredicted Paths," Workshop on Performance Analysis and its Impact on Design (PAID), Barcelona, Spain, 1998.
- P. Bose, J.A. Abraham,"Performance and Functional Verification of Microprocessors," Proc. 13th IEEE International Conference on VLSI Design, January 2000.
- P. Bose,"Performance Test Case Generation for Microprocessors,"Proc. 16th IEEE VLSI Test Symposium, April 1998, pp. 54-59.
- P. Bose et al., "Bounds-Based Loop Performance Analysis: Application to Validation and Tuning,"Proc. IEEE Int'l Performance, Computing and Communications Conference, February 1998, pp. 178-184.
- Other documents in doc/ in the Turandot package.

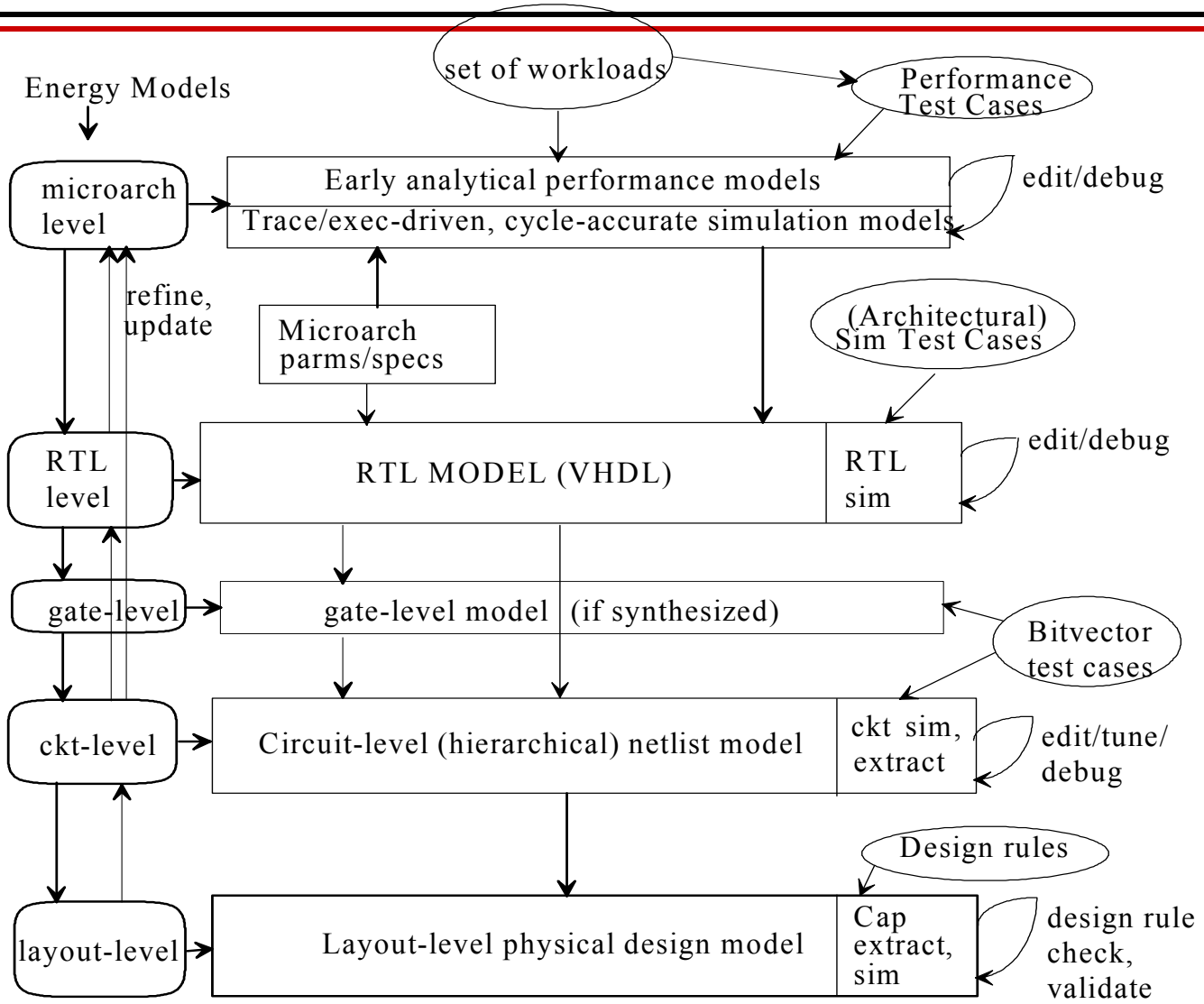
Tutorial Outline

9:00-10:30	Introduction and Motivation Basics of Performance Modeling - Turandot performance simulation infrastructure Architectural Power Modeling - PowerTimer extensions to Turandot - Power-Performance Efficiency Metrics Case Studies and Examples - Optimal Power-Performance Pipeline Depth Validation and Calibration Efforts Future challenges and Discussion Bibliography

Dynamic Power Estimation



Modeling Hierarchy and Tool Flow



Architecture level models

Power ~ ½ CV^2Af

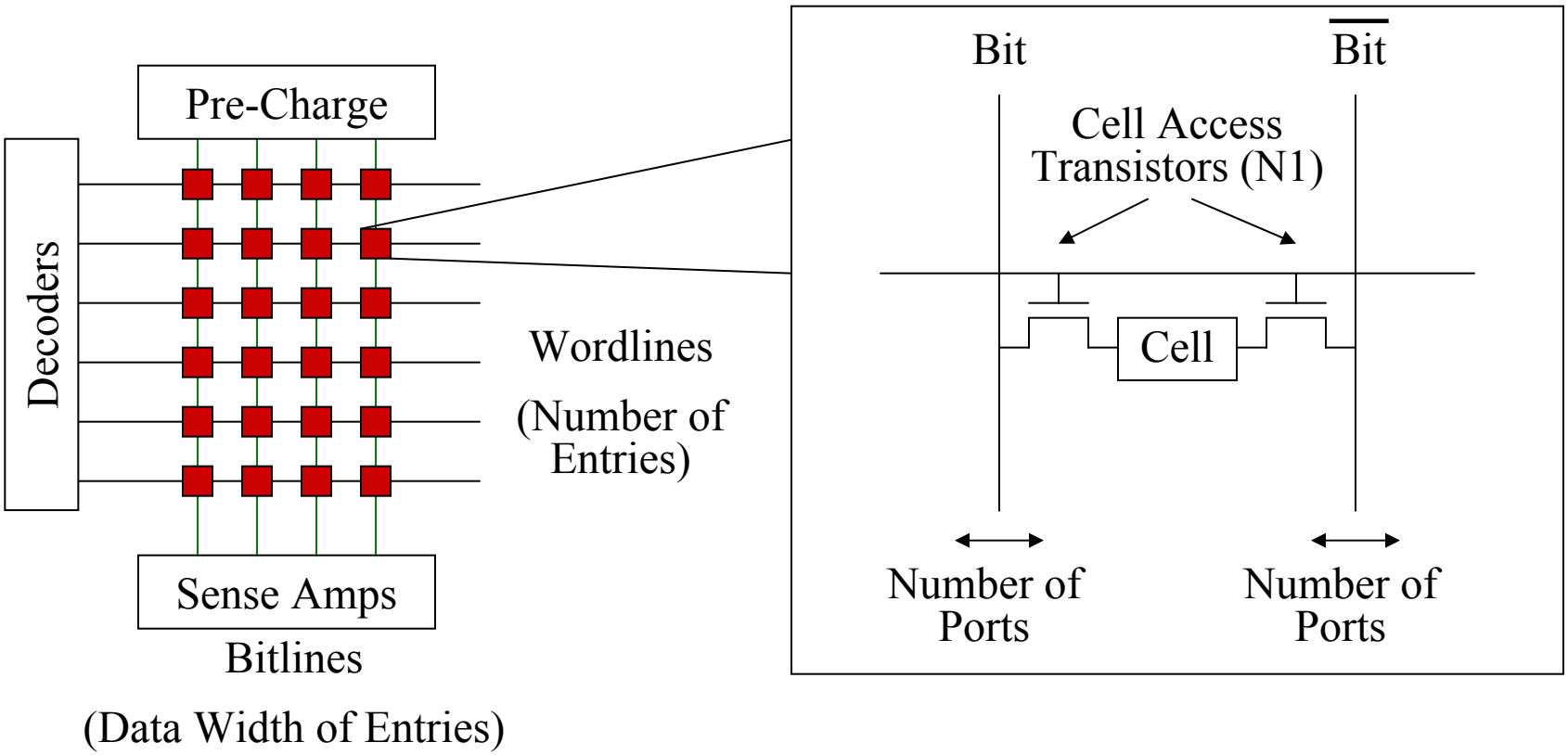
- Bottom-up Approach:
  - Estimate "CV^2f" via analytical models
  - Tools: Wattch, PowerAnalyzer, Tempest (mixed-mode)
- Top-Down Approach
  - Estimate "CV^2f" via empirical measurements
  - Tools: PowerTimer, AccuPower, Internal Industrial Tools

- Estimate "A" via statistics from architectural-performance simulators

Analytical Modeling Tools: Modeling Capacitance

- Requires modeling wire length and estimating transistor sizes
- Related to RC Delay analysis for speed along critical path
  - But capacitance estimates require summing up all wire lengths, rather than only an accurate estimate of the longest one.

Register File: Capacitance Analysis

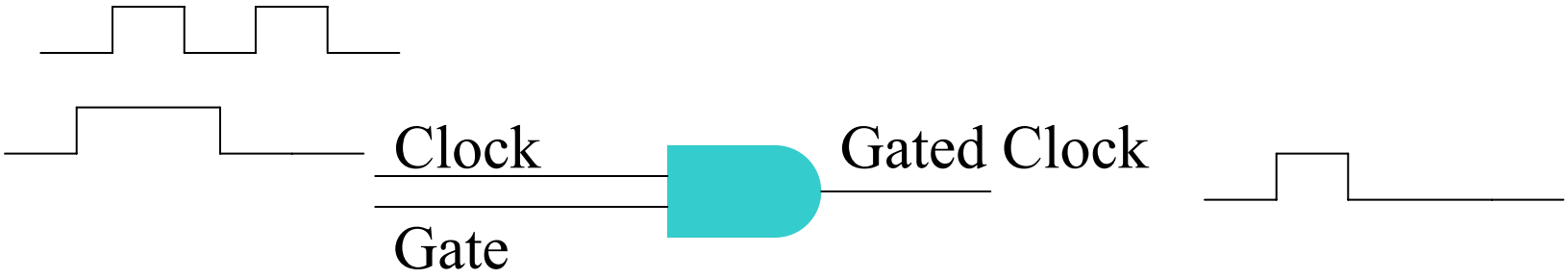


$$C_{bitline} = C_{diffcapPchg} + NumberWordlines * C_{diffcapN1} + Bitlinelength * C_{metal}$$

Architecture level models: Signal Transition Statistics

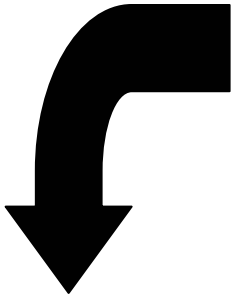
- Dynamic power is proportional to switching
- How to collect signal transition statistics in architectural-level simulation?
  - Many signals are available, but do we want to use all of them?
  - One solution (register file):
    - Collect statistics on the important ones (bitlines)
    - Infer where possible (wordlines)
    - Assign probabilities for less important ones (decoders)

Architecture level models:
Clock Gating: What, why, when?



- Dynamic Power is dissipated on clock transitions
- Gating off clock lines when they are unneeded reduces activity factor
- But putting extra gate delays into clock lines increases clock skew
- End results:
  - Clock gating complicates design analysis but saves power.

Wattch: An Overview



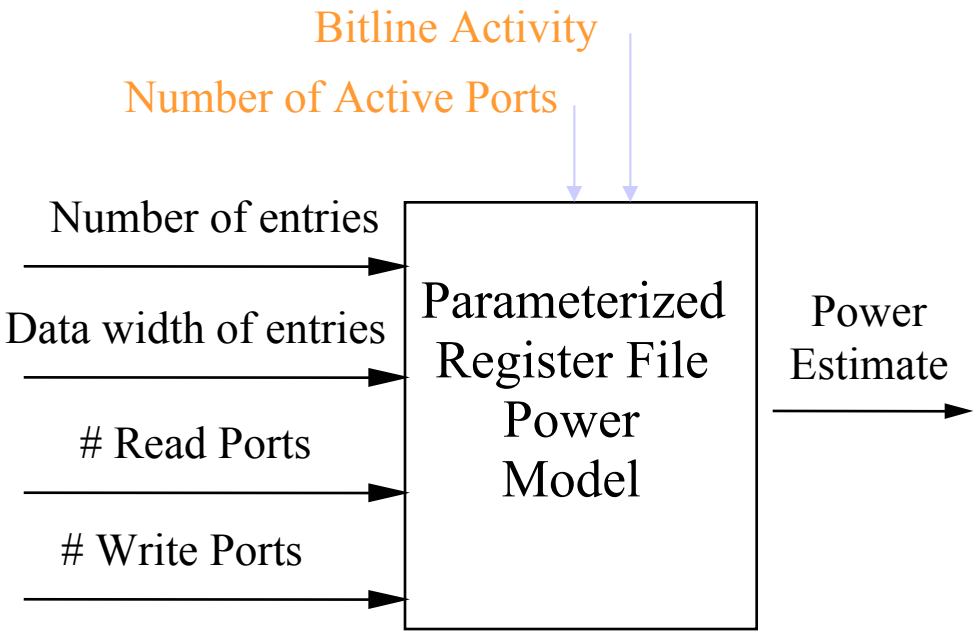
Wattch's Design Goals

- Flexibility
- Planning-stage info
- Speed
- Modularity
- Reasonable accuracy

Overview of Features

- Parameterized models for different CPU units
  - Can vary size or design style as needed
- Abstract signal transition models for speed
  - Can select different conditional clocking and input transition models as needed
- Based on SimpleScalar (has been ported to many simulators)
- Modular: Can add new models for new units studied

Unit Modeling



Modeling Capacitance

- Models depend on structure, bitwidth, design style, etc.
- E.g., may model capacitance of a register file with bitwidth & number of ports as input parameters

Modeling Activity Factor

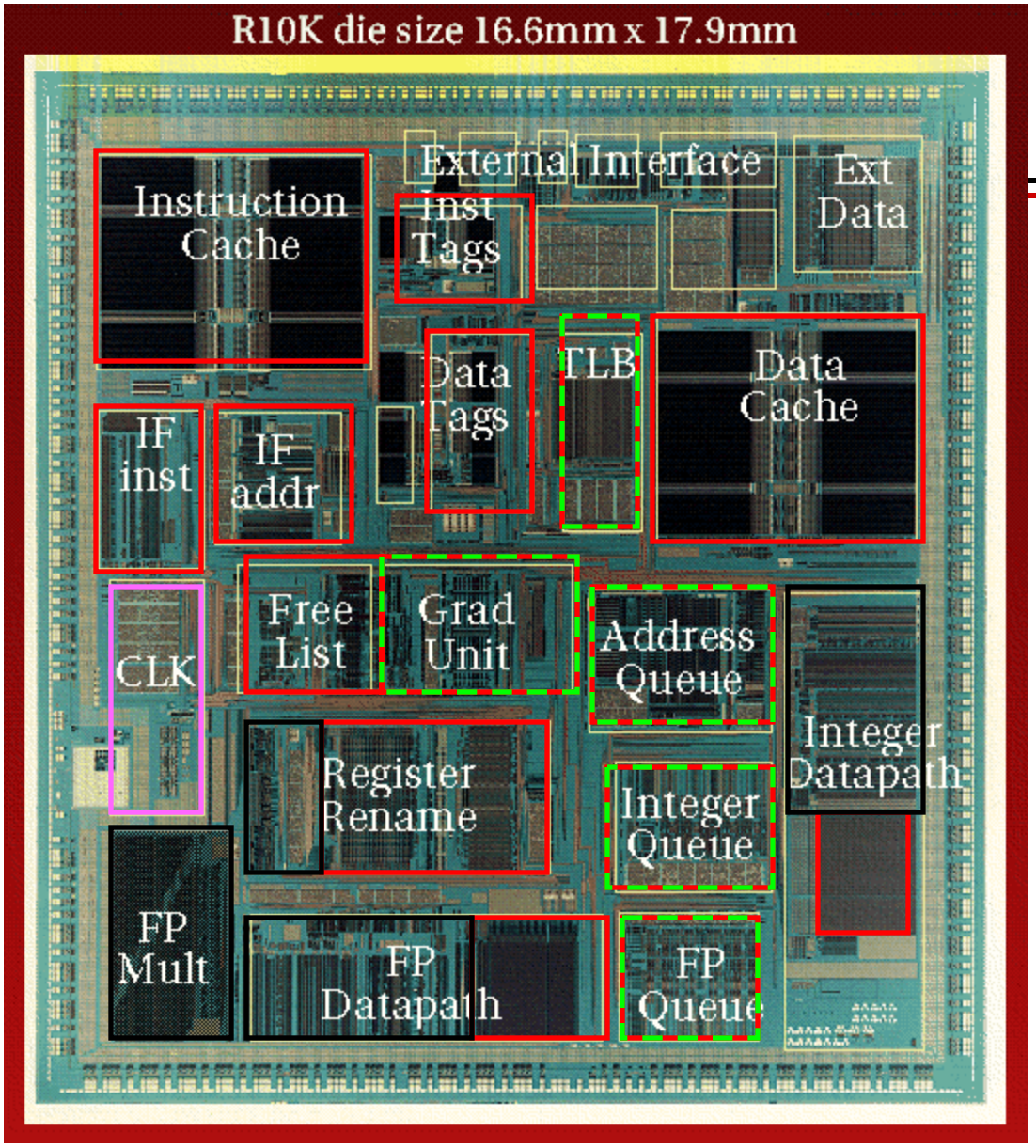
- Use cycle-level simulator to determine number and type of accesses
  - reads, writes, how many ports
- Abstract model of bitline activity

One Cycle in Wattch

	Fetch	Dispatch	Issue/Execute	Writeback/ Commit
Power (Units Accessed)	• I-cache • Bpred	• Rename Table • Inst. Window • Reg. File	• Inst. Window • Reg File • ALU • D-Cache • Load/St Q	• Result Bus • Reg File • Bpred
Performance	• Cache Hit? • Bpred Lookup?	• Inst. Window Full?	• Dependencies Satisfied? • Resources?	• Commit Bandwidth?

- On each cycle:
  - determine which units are accessed
  - model execution time issues
  - model per-unit energy/power based on which units used and how many ports.

Units Modeled by Wattch

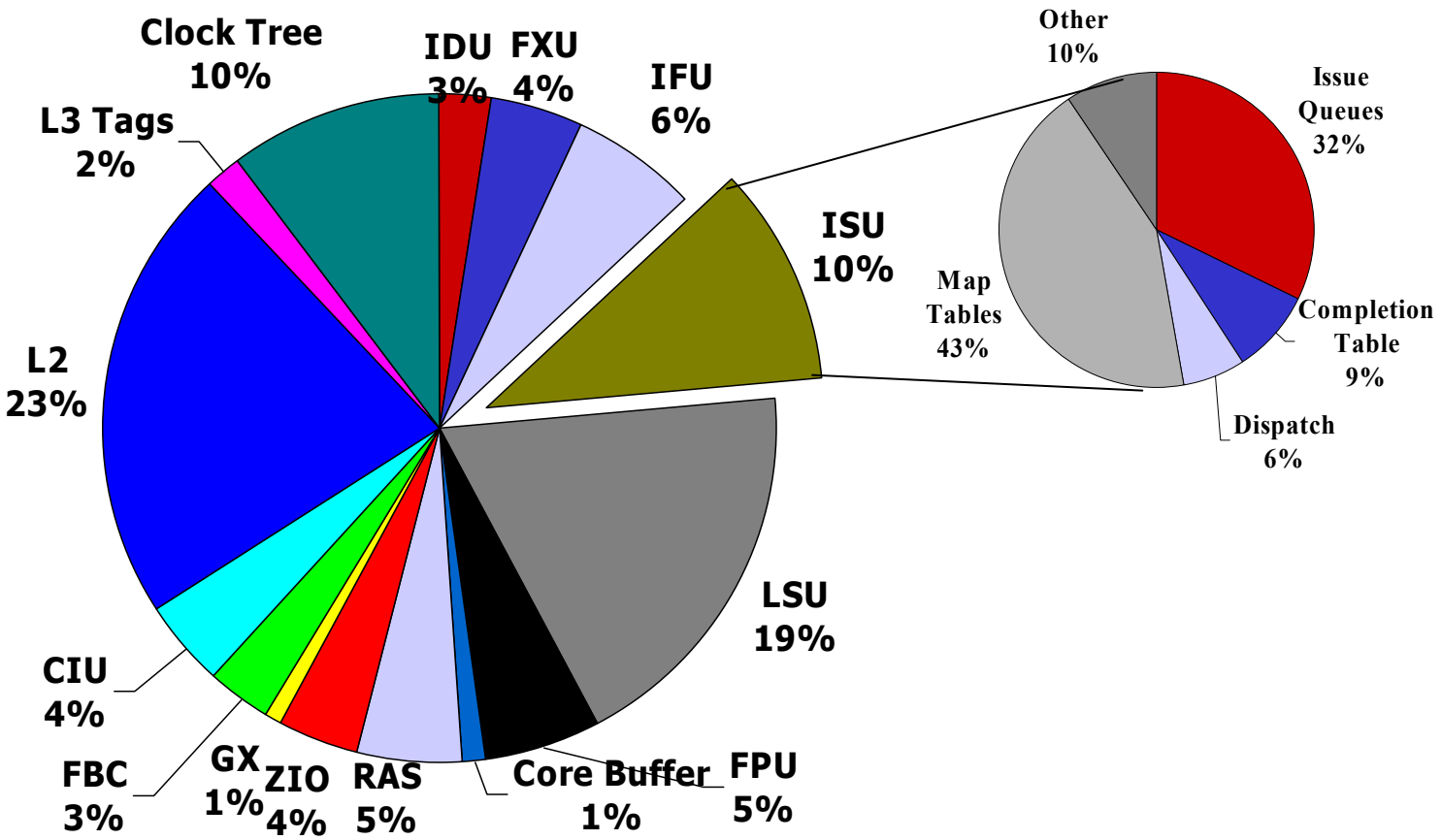


- **Array Structures**
  - Caches, Reg Files, Map/Bpred tables
- **Content-Addressable Memories (CAMs)**
  - TLBs, Issue Queue, Reorder Buffer
- **Complex combinational blocks**
  - ALUs, Dependency Check
- **Clocking network**
  - Global Clock Drivers, Local Buffers

PowerTimer

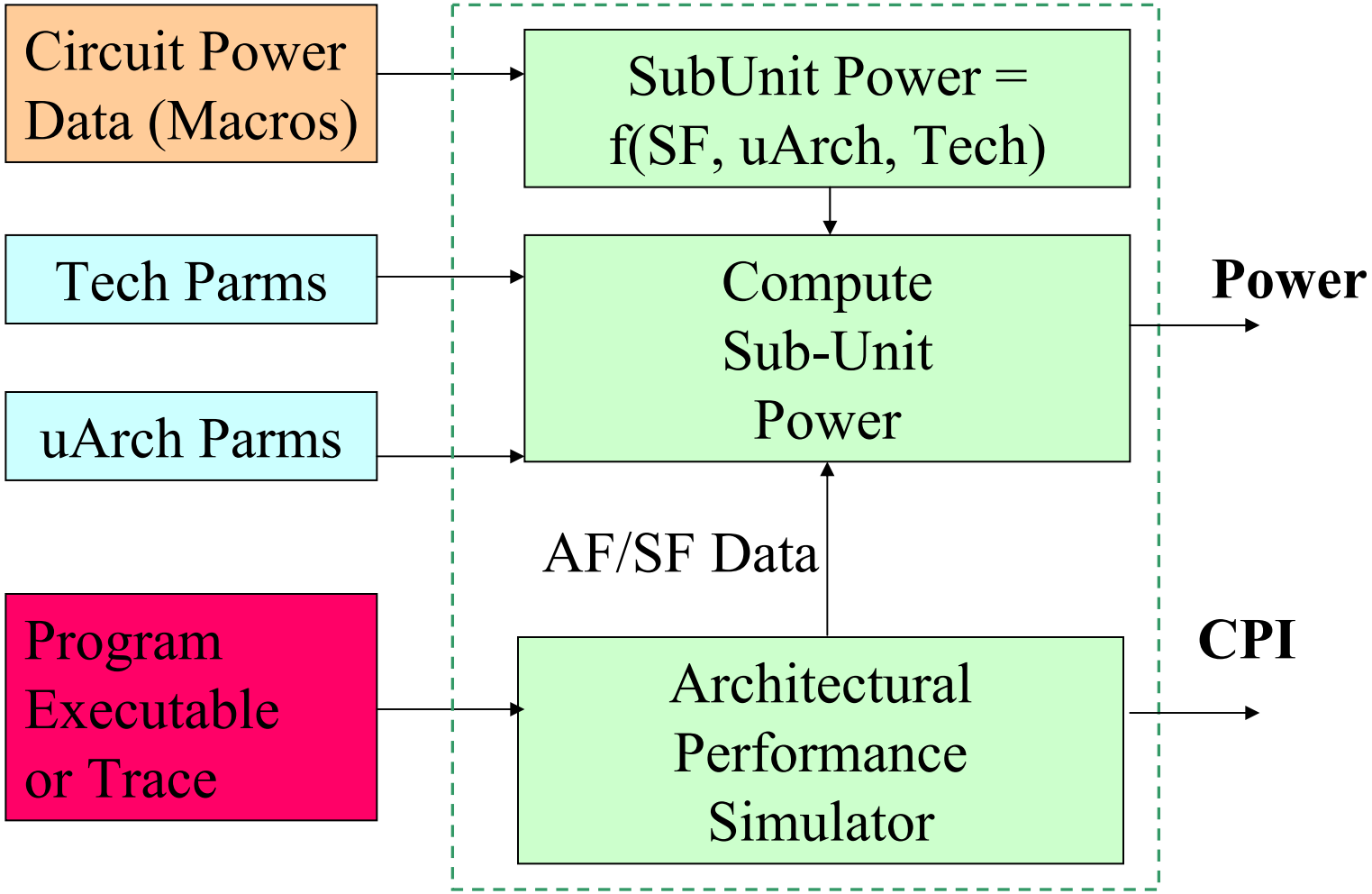
- IBM Tool First Develop During Summer of 2000
  - Continued Development: 2001 => Today
  - Methodology Applied to Research and Product Power-Performance Simulators with IBM
  - Currently in Beta-Release
  - Working towards Full Academic Release

PowerTimer: Empirical Unconstrained Power



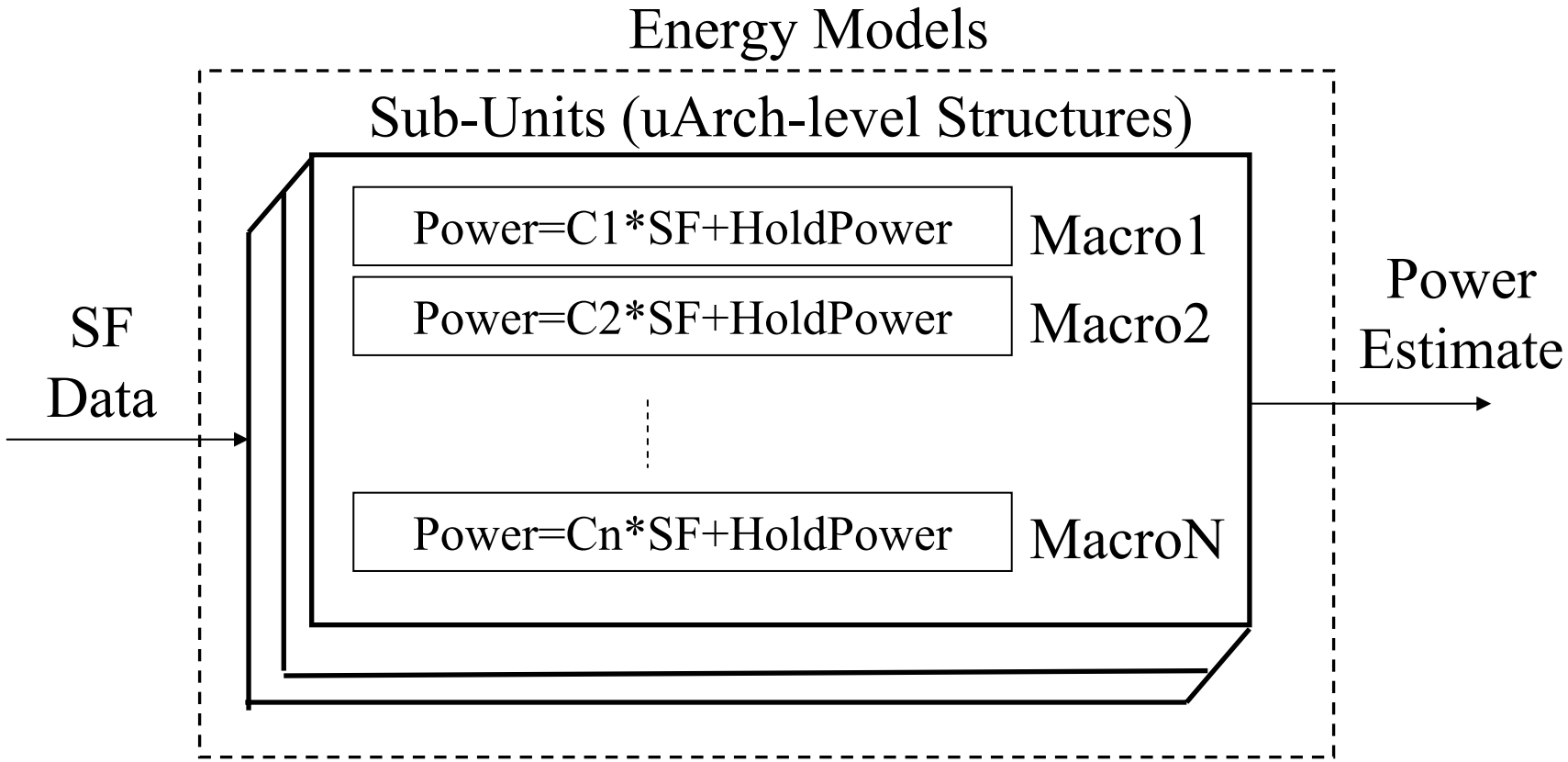
Pre-silicon, POWER4-like superscalar design

PowerTimer


Circuit Power Data (Macros)
SubUnit Power = f(SF, uArch, Tech)
Tech Parms
Compute Sub-Unit Power
Power
uArch Parms
AF/SF Data
Program Executable or Trace
Architectural Performance Simulator
CPI

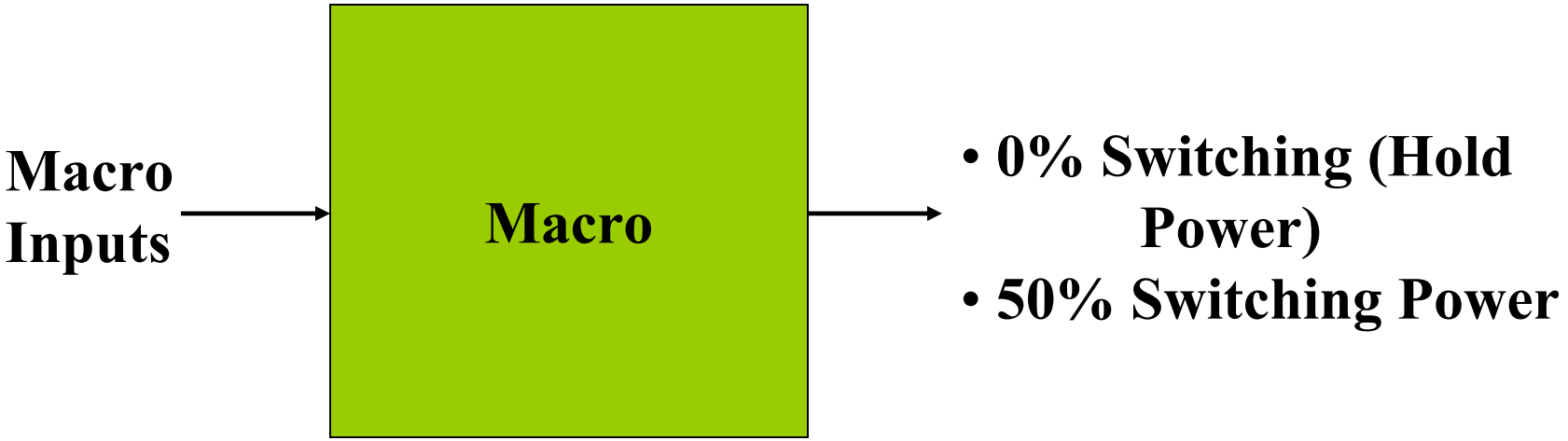
PowerTimer: Energy Models

- Energy models for uArch structures formed by summation of circuit-level macro data



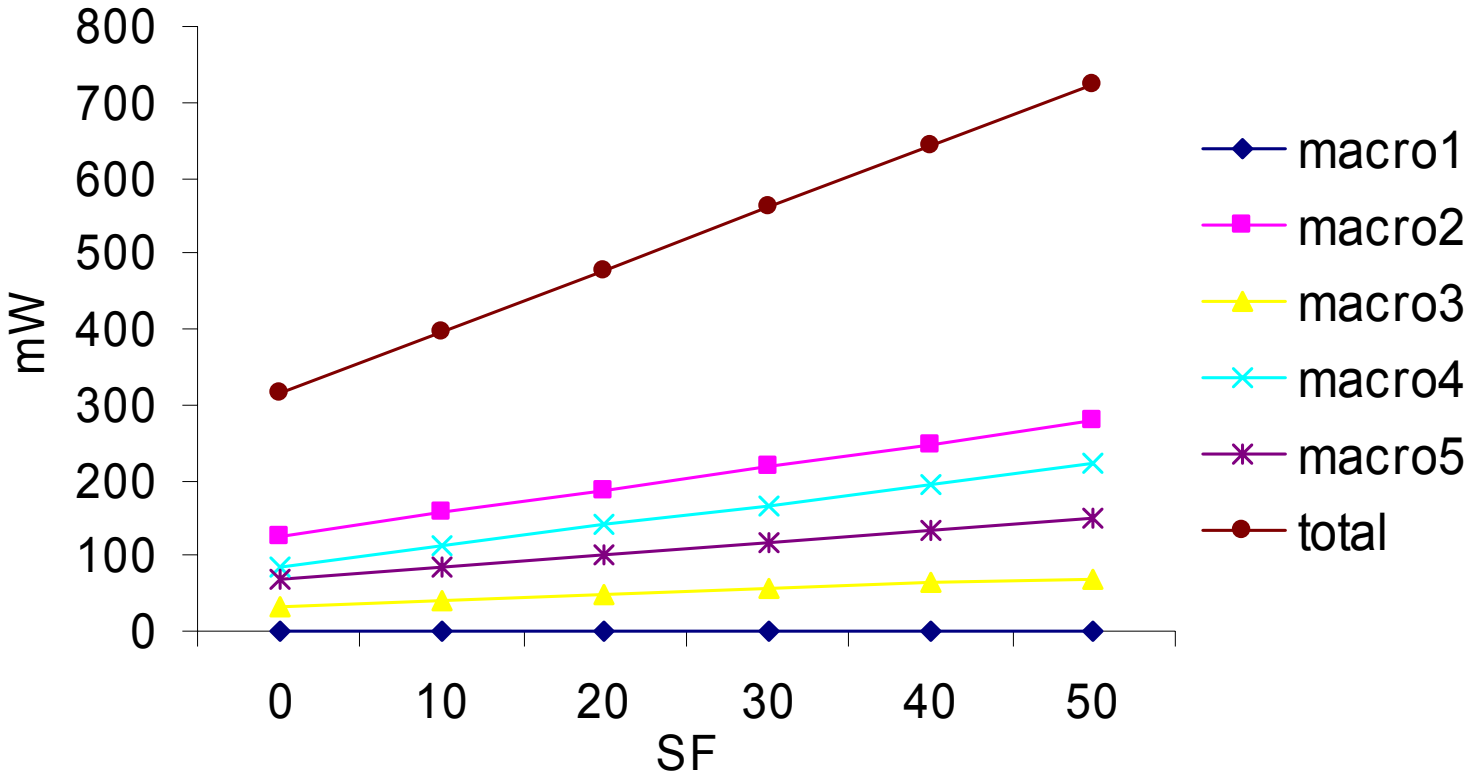
Empirical Estimates with CPAM

- Estimate power under "Input Hold" and "Input Switching" Modes
- Input Hold: All Macro Inputs (Except Clocks) Held
  - Can also collect data for Clock Gate Signals
- Input Switching: Apply Random Switching Patterns with 50% Switching on Input Pins



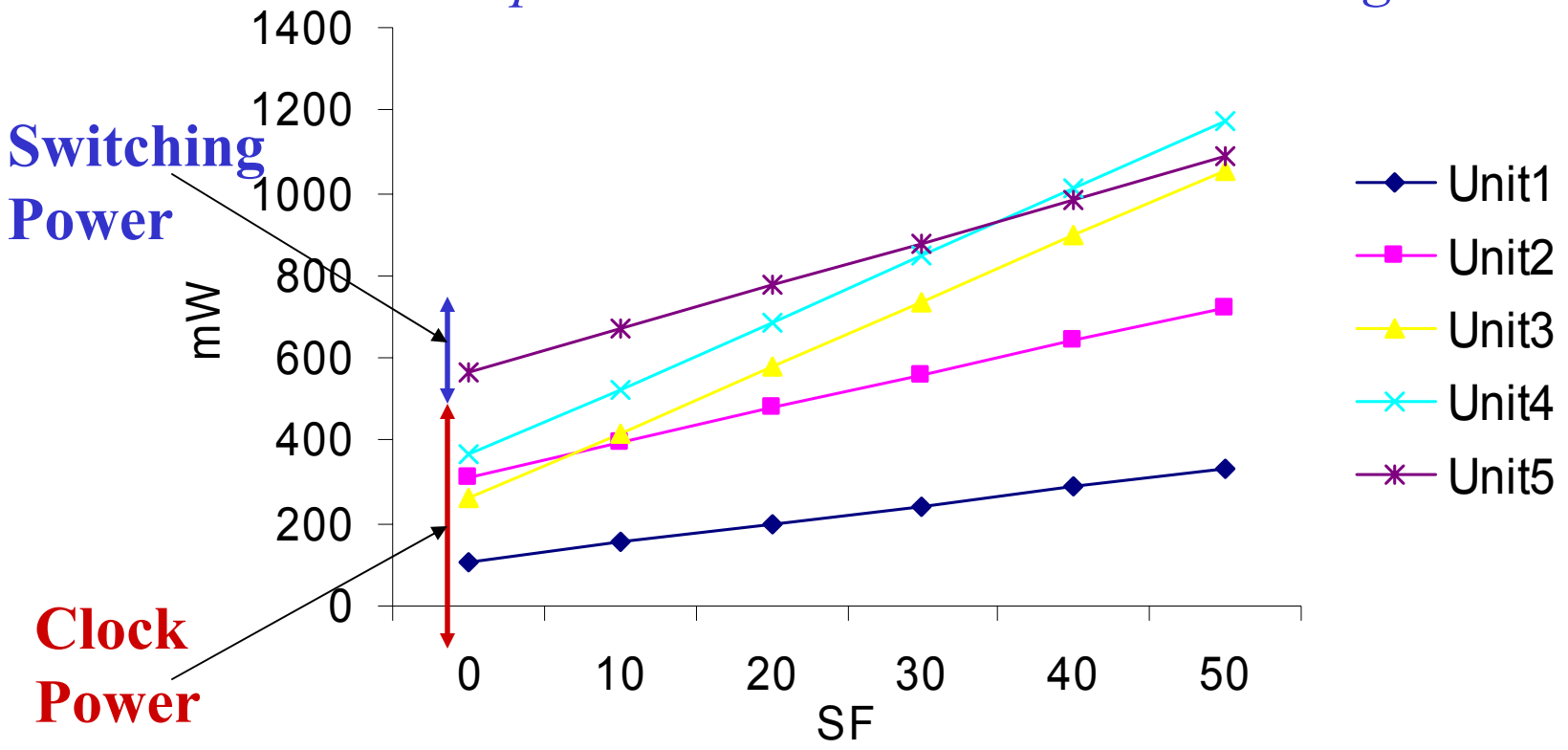
A Sample Unit

- Made up of 5 macros
  - macro1, macro2, macro3, macro4, macro5



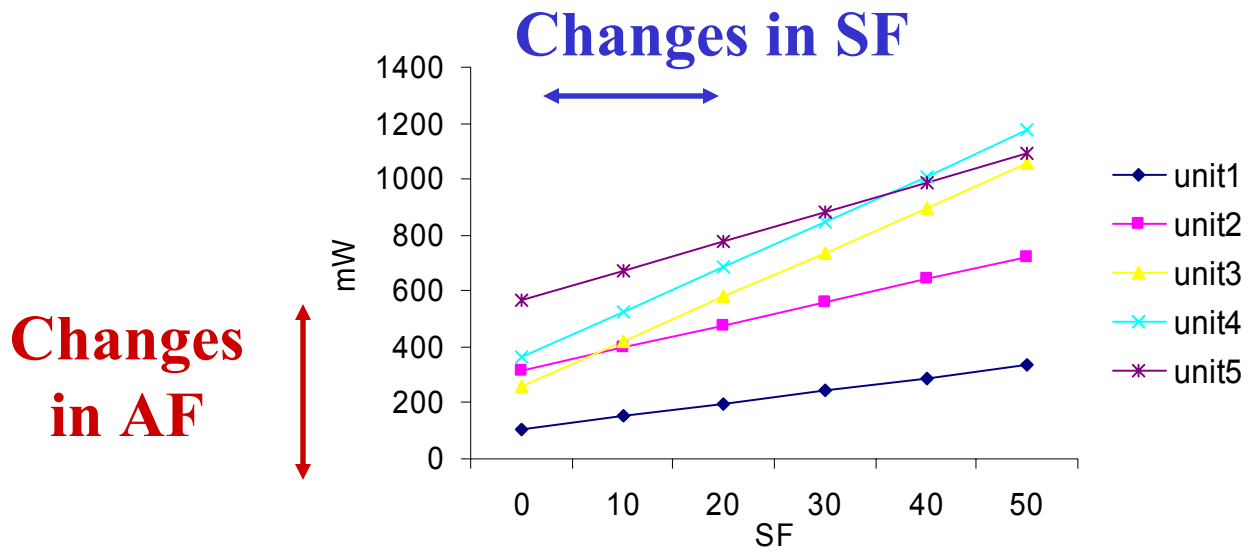
PowerTimer: Power models f(SF)

Assumption: Power linearly dependent on Switching Factor
This separates Clock Power and Switching Power



At 0% SF, Power = Clock Power (significant without clock gating)

Key Activity Data



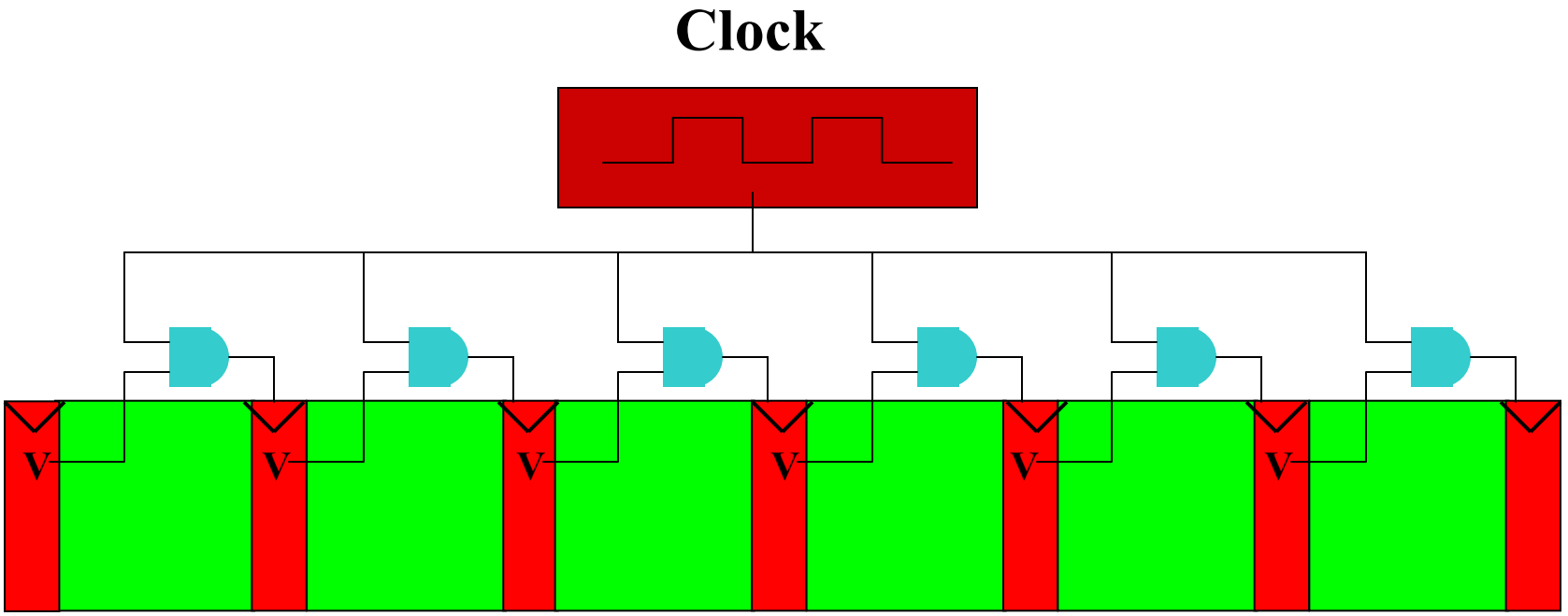
- SF => Moves along the Switching Power Curve
  - Estimated on a per-unit basis from RTL Analysis
- AF => Moves along the Clock Power Curve
  - Extracted from Microarchitectural Statistics (Turandot)

Microarchitectural Statistics

- Stats are very similar to tracking used in Wattch, etc
- Differences:
  - Clock Gating Modes (3 modes)
  - Customized Scaling Based on Circuit Style (4 styles)
- Clock Gating Modes:
  - P_constrained = P_unconstrained (not clock-gateable)
  - P_constrained_1 = AF * (P_{clock} + P_{logic}) (common)
  - P_constrained_2 = AF * P_{clock} + P_{logic} (rare)
  - P_constrained_3 = P_{clock} + AF * P_{logic} (very rare)
- Scaling Based on Circuit Styles
  - AF_1 = #valid (Latch-and-Mux, No Stall Gating)
  - AF_2 = #valid - #stalls (Latch-and-Mux, With Stall Gating)
  - AF_3 = #writes (Arrays that only gate updates)
  - AF_4 = #writes + #reads (Arrays, RAM Macros)

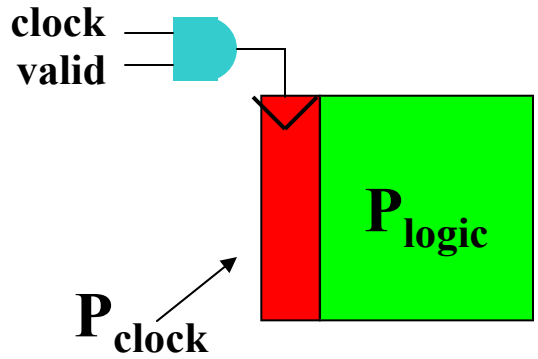
Clock Gating Modes: Valid-Bit Gating

- Latch-Based Structures: Execute Pipelines, Issue Queues

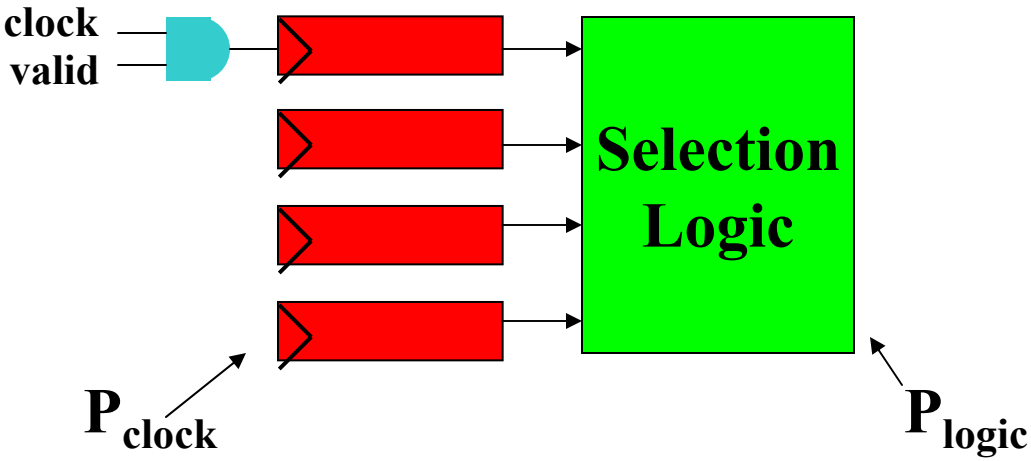


Clock Gating Modes

- P_constrained_1 = AF * (P_{clock} + P_{logic})

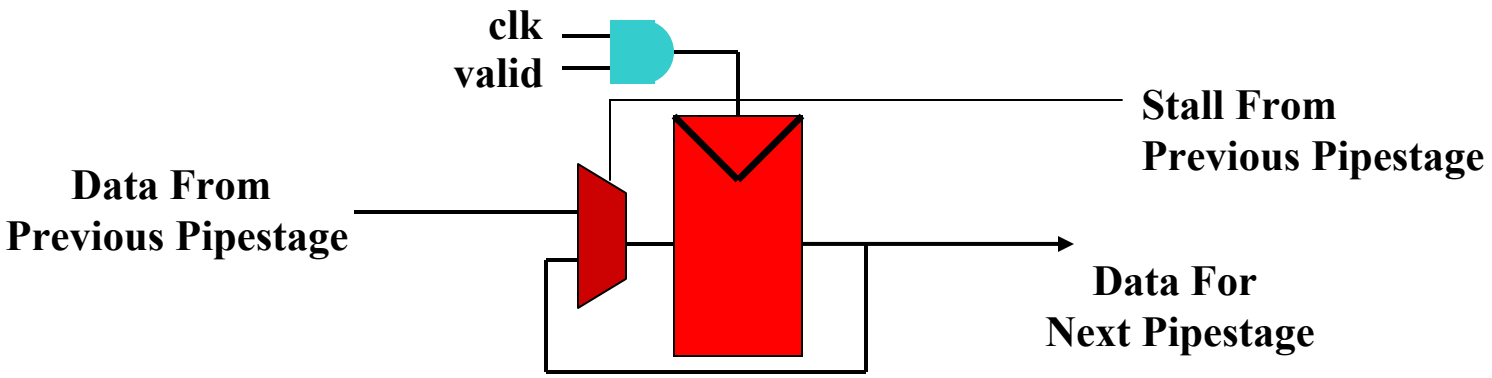


- P_constrained_2 = AF * P_{clock} + P_{logic}

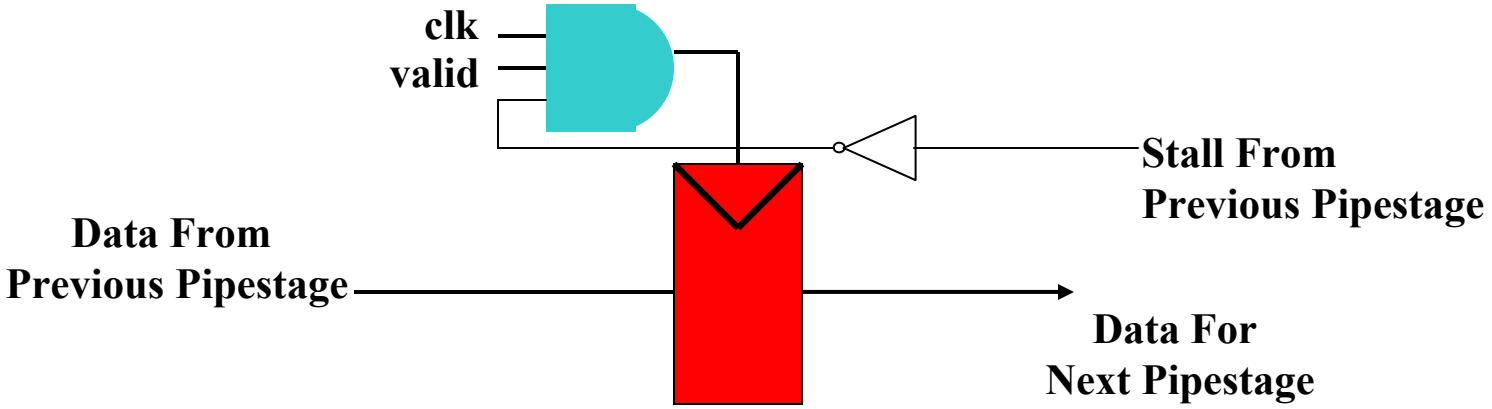


Scaling Options: Valid-bit Gating, what about Stalls?

- Option 1: Stalls *cannot* be gated

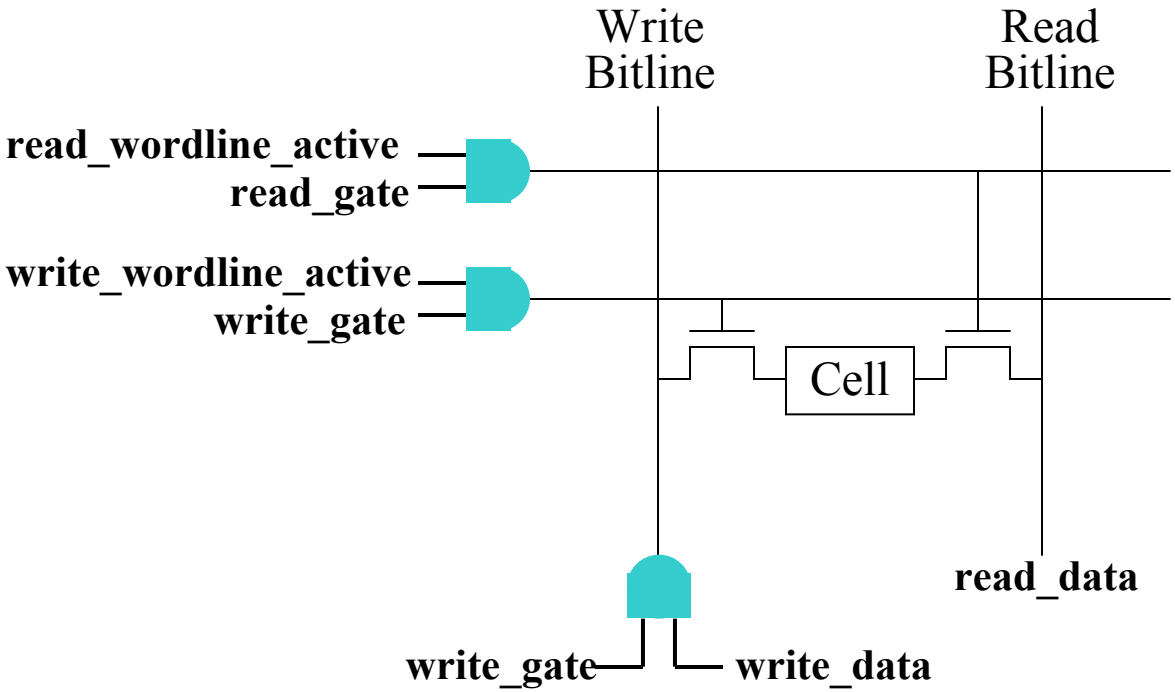


- Option 2: Stalls *can* be gated



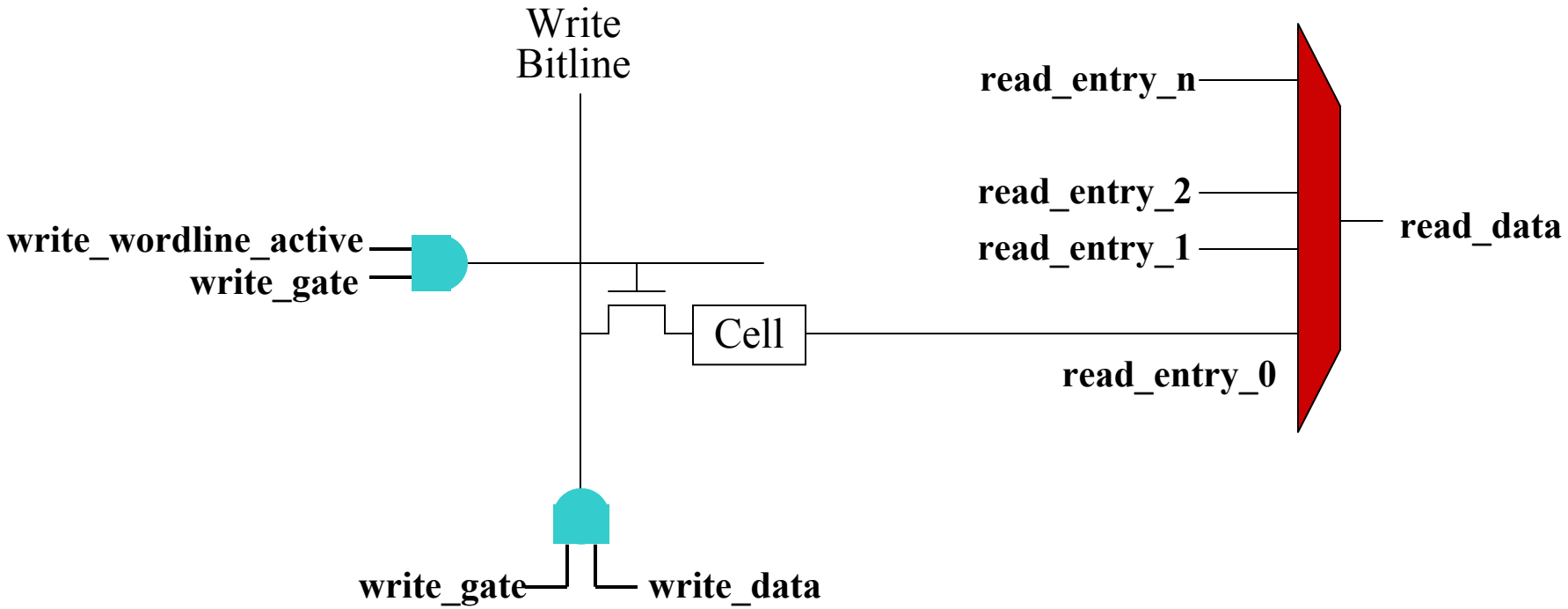
Scaling Options: Array Structures

- Option 1: Reads and Writes Eligible to Gate for Power



Scaling Options: Array Structures

- Option 2: Only Writes Eligible to Gate for Power



12 Clock Gating Modes

Gating Mode	Valid	Valid+ Stalls	Writes	Writes+ Reads	Gate Both	Gate Clock	Gate Logic	Examples
0	No	No	No	No	No	No	No	Control Logic, Buffers, Small Macros
1	Yes	No	No	No	Yes	No	No	Issue Queues, Execute Pipelines
2	No	Yes	No	No	Yes	No	No	
3	No	No	Yes	No	Yes	No	No	Caches
4	No	No	No	Yes	Yes	No	No	Some Queues
5	Yes	No	No	No	No	Yes	No	CAMs, Selection Logic
6	No	Yes	No	No	No	Yes	No	
7	No	No	Yes	No	No	Yes	No	No Known macros
8	No	No	No	Yes	No	Yes	No	No Known macros
9	Yes	No	No	No	No	No	Yes	No Known macros
10	No	Yes	No	No	No	No	Yes	No Known macros
11	No	No	Yes	No	No	No	Yes	No Known macros
12	No	No	No	Yes	No	No	Yes	No Known macros

PowerTimer Observations

- PowerTimer works well for POWER4-like estimates and derivatives
  - Scale base microarchitecture quite well
  - E.g. optimal power-performance pipelining study
  - Lack of run-time, bit-level SF not seen as a problem within IBM (seen as noise)
    - Chip bit-level SFs are quite low (5-15%)
    - Most (60-70%) power is dissipated while maintaining state (arrays, latches, clocks)
    - Much state is not available in early-stage timers

Comparing models: Flexibility

- Flexibility necessary for certain studies
  - Resource tradeoff analysis
  - Modeling different architectures
- Purely analytical tools provides fully-parameterizable power models
  - Within this methodology, circuit design styles could also be studied
- PowerTimer scales power models in a user-defined manner for individual sub-units
  - Constrained to structures and circuit-styles currently in the library
- Perhaps Mixed Mode tools could be very useful

Comparing power models: Accuracy

- PowerTimer -- Based on validation of individual pieces
  - Extensive validation of the performance model (AFs)
  - Power estimates from circuits are accurate
  - Circuit designers must vouch for clock gating scenarios
  - Certain assumptions will limit accuracy or require more in-depth analysis
- Analytical Tools
  - Inherent Issues
    - Analytical estimates cannot be as accurate as SPICE analysis ("C" estimates, CV^2 approximation)
  - Practical Issues
    - Without industrial data, must estimate transistor sizing, bits per structure, circuit choices

Comparing power models: Speed

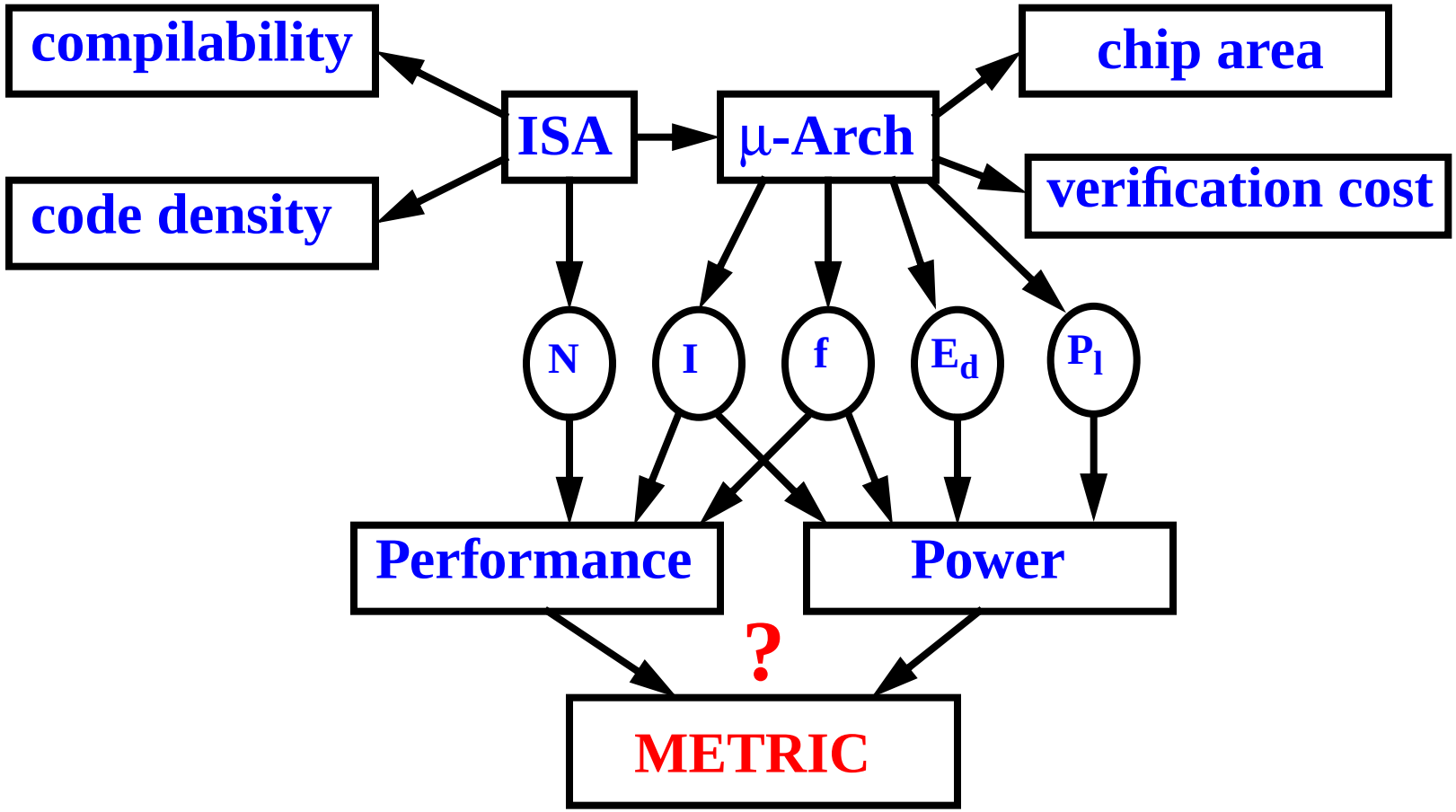
- Performance simulation is slow enough!
- Post-Processing vs. Run-Time Estimates
- Wattch's per-cycle power estimates: roughly 30% overhead
  - Post-processing (per-program power estimates) would be much faster (minimal overhead)
- PowerTimer allows both no overhead post-processing and run-time analysis for certain studies (di/dt, thermal)
  - Some clock gating modes may require run-time analysis
- Third Option: Bit Vector Dumps
  - Flexible Post-Processing ⇔ Huge Output Files

Bibliography: Architectural Power Modeling

- David Brooks, Vivek Tiwari, and Margaret Martonosi. "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," 27th International Symposium on Computer Architecture (ISCA), Vancouver, British Columbia, June 2000.
- David Brooks, John-David Wellman, Pradip Bose, and Margaret Martonosi. "Power-Performance Modeling and Tradeoff Analysis for a High-End Microprocessor," Workshop on Power-Aware Computer Systems (PACS2000, held in conjuction with ASPLOS-IX), Cambridge, MA., November, 2000.
- J. Scott Neely, Howard H. Chen, Steven G. Walker, James Venuto, and Thomas J. Bucelot, "CPAM: A Common Power Analysis Methodology for High-Performance VLSI Design," 9th Topical Meeting on Electrical Performance of Electronic Packaging, Oct. 23-25, 2000, Scottsdale, AZ.
- J. D. Warnock, J. M. Keaty, J. Petrovick, J. G. Clabes, C. J. Kircher, B. L. Krauter, P. J. Restle, B. A. Zoric, and C. J. Anderson, “The circuit and physical design of the POWER4 microprocessor,” IBM Journal of Research and Development, Volume 46, No. 1, 2002.
- David Brooks, Pradip Bose, Viji Srinivasan, Michael Gschwind, Philip G. Emma, Michael G. Rosenfield. "New methodology for early-stage, microarchitecture-level power-performance analysis of microprocessors," IBM Journal of Research and Development, Volume 47, No. 5/6, 2003.

Power-Performance Metric for Optimizing Architecture

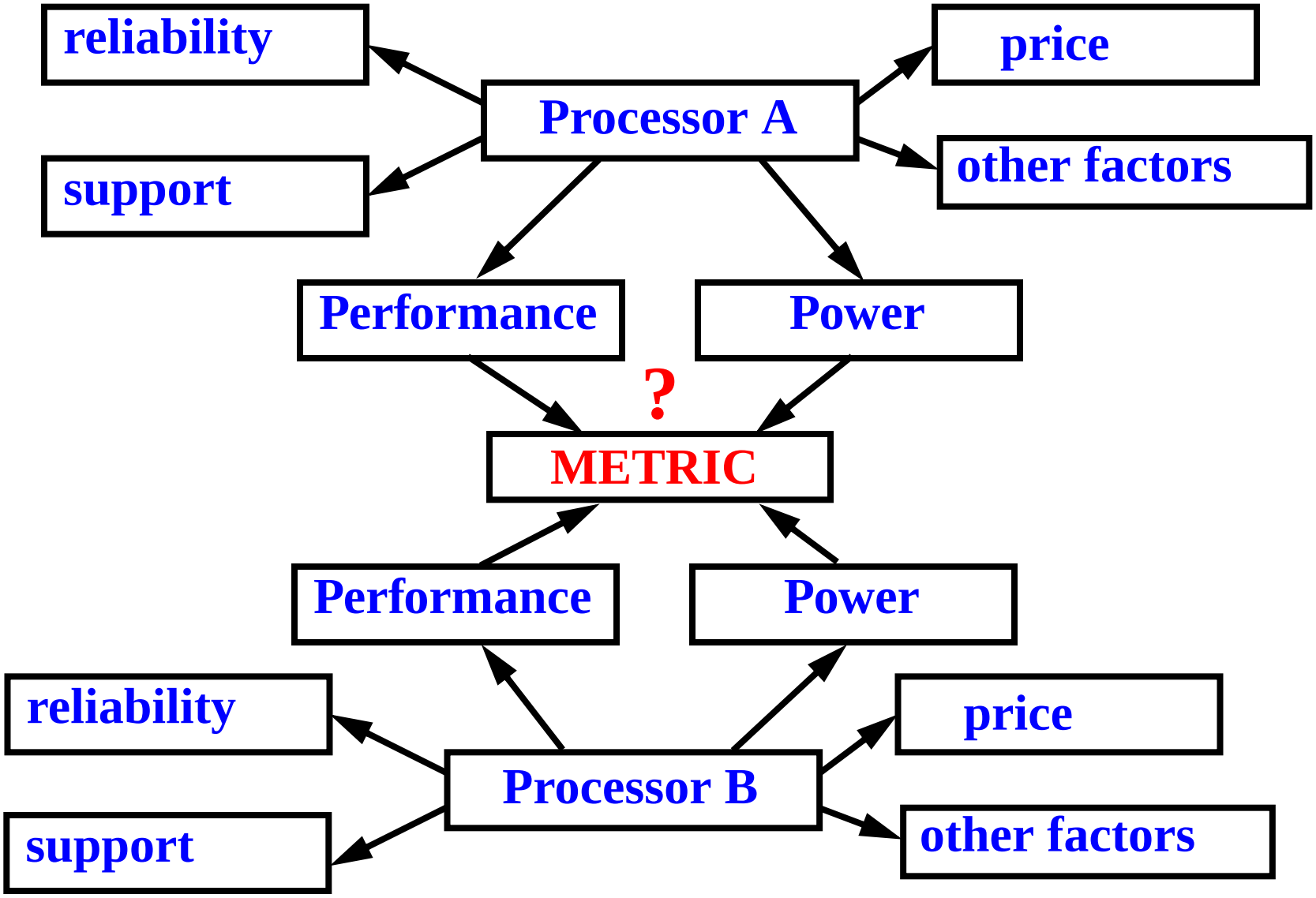
Customer should not necessarily use this metric



N	**dynamic instr. count**	**P_l**	**leakage power**	**f**	**max frequency**
I	**architectural speed, IPC**	**E_d**	**average switching energy**		

End-User Power-Performance Metric

Customer does not need to know circuits, or implementation details to choose product



Existing Power-Performance Metric

- $\frac{\text{BIPS}}{\text{Watt}}$, reverse of "energy per operation"
  - used for comparing low-end products
  - incorrectly used for "fixed throughput mode" and "power-limited mode"
- $\frac{\text{BIPS}^2}{\text{Watt}}$, reverse of "energy-delay product"
  - used for comparing mid-range products
- $\frac{\text{BIPS}^3}{\text{Watt}}$, "*Vdd* - invariant" metric (neglecting leakage)
  - assumes $\text{BIPS} \propto f$, $f \propto \text{Vdd}$ (around nominal *Vdd*), $E_{\text{dynamic}} \propto \text{Vdd}^2$
  - changing power supply (within some limits) does not change the metric
- $\frac{\text{BIPS}^\gamma}{\text{Watt}}$, with more creative methods for determining γ
  - $\gamma > 3$ for comparing products with emphasis on performance
  - in leakage-dominated designs the "*Vdd* - invariant" metric leads to
    - $\gamma > 3$ if $P_{leakage}$ grows faster than Vdd^3 around nominal *Vdd*
    - $\gamma < 3$ if $P_{leakage}$ grows slower than Vdd^3 around nominal *Vdd*

Existing Power-Performance Metric (cont.)

- $\frac{\text{BIPS}^{\gamma}}{\text{Watt}}$, may be useful for theoretical analysis or for marketing,
  but not easy to use for optimizing architecture (evaluating architectural features)
  - metric is correct, but using it may be confusing (hides important assumptions)
  - may not know BIPS or Watt early in design
  - may attempt to estimate ΔBIPS and ΔWatt, but
    - although architects know how to estimate ΔIPC, it is much more difficult to predict Δf
    - designs under evaluation have to be properly tuned before applying the metric
    - ΔWatt is difficult to estimate because it requires knowing Δf, $\Delta E_{dynamic}$ and $\Delta P_{leakage}$
    - to measure $\Delta E_{dynamic}$ and $\Delta P_{leakage}$ pipeline needs to be retuned, depending on assumption about Δf

- Recently introduced "unified metric" $\Theta\frac{\Delta IPC}{IPC} > \frac{\Delta E}{E} + \Sigma\eta_i w_i \frac{\Delta D_i}{T} + (\Theta + 1)\frac{\Delta N}{N}$,

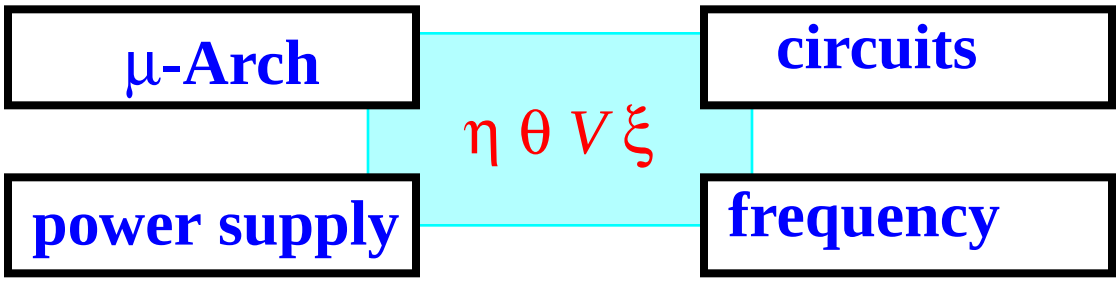
  V. Zyuban, GLSVLSI, 04/18/2002
  V. Zyuban and P. Strenski, ISLPED, 08/12/2002
  V. Zyuban and P. Strenski, IBM JRD, Dec. 2003
  - use for optimizing architecture (not suitable as a end-user metric)

- Special case $3\frac{\Delta IPC}{IPC} > \frac{\Delta Power}{Power}$

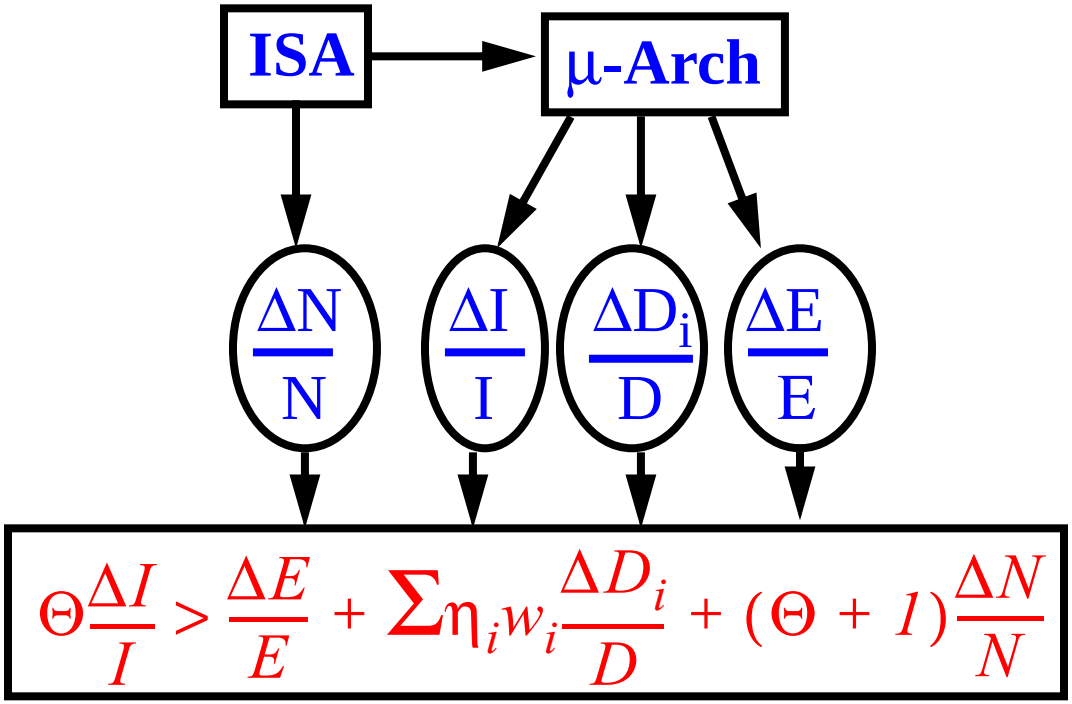
  J. Rattner, MICRO-35 keynote speech, 11/22/2002

Unified Energy-Efficiency Metric - Fundamentals

- Techniques power and performance
  - system: raise *Vdd*
  - technology: shrink Tox or Leff
  - circuits: increase device sizes, restructure logic to increase parallelism
  - microarchitecture: deepen the pipeline (reduce FO4), increase issue width, add functions, bypasses, increase the number of ports and entries in queues and register files, build more aggressive caches, branch predictors, etc.
- Main idea: balance design decisions across all domains
  - in particular, balance architectural decisions with technology and circuit-level choices
- All costs are measurable - introduce variables to quantify the tradeoffs
  - hardware intensity η
  - voltage intensity Θ
  - architectural intensity ξ



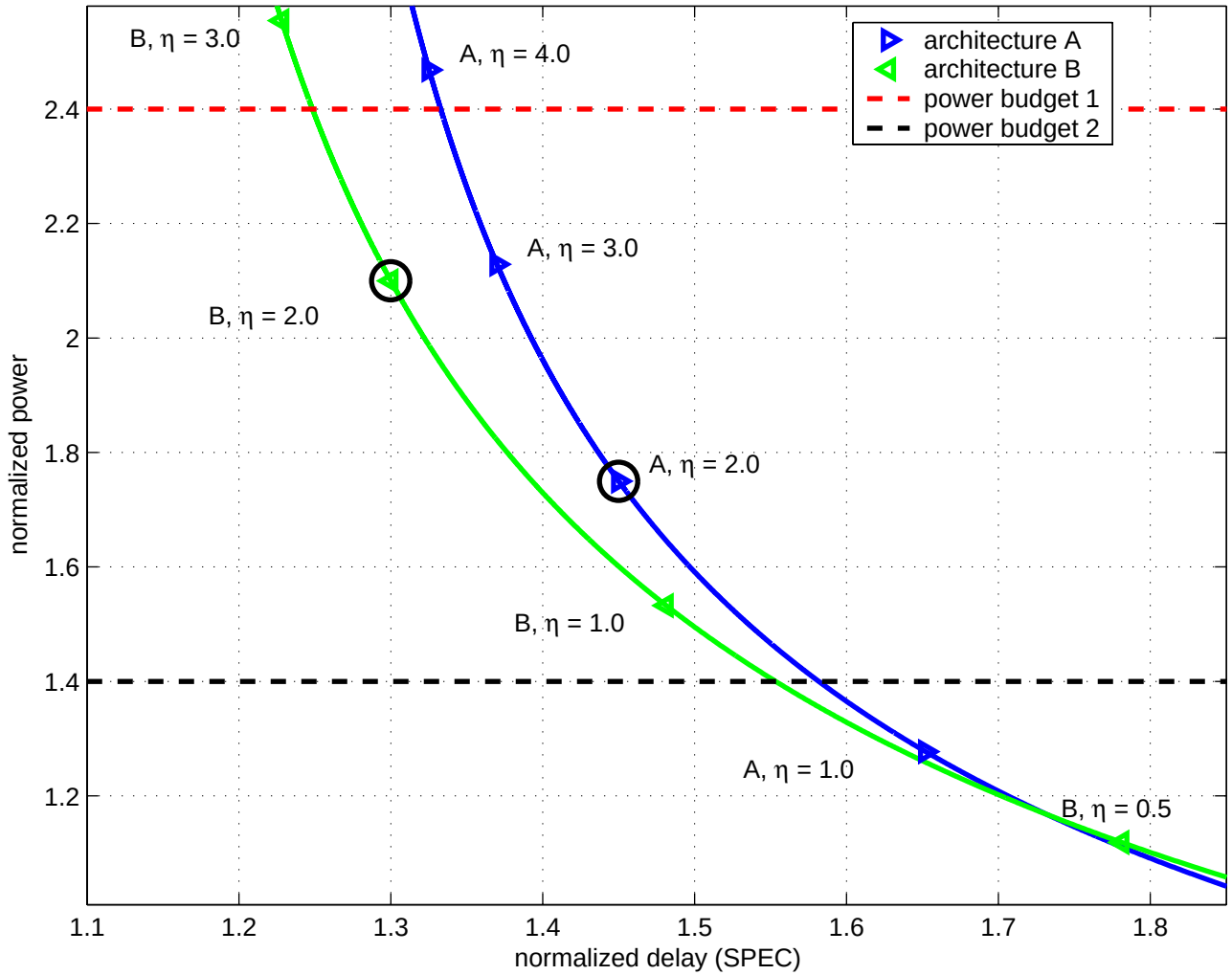
Unified Energy-Efficiency Metric



All parameters have clear physical meaning, and a method for measuring them.

I	**architectural speed, IPC**	**Θ**	**depends on technology and Vdd**
E	**average energy per instruction**	**η_i**	**hardware intensity in stage i**
N	**dynamic instruction count**	**w_i**	**energy weight of stage i**
D_i	**critical path delay through stage i**	**Δ 's**	**'naive' increments (no retuning)**

Unified Metric - Graphical Interpretation



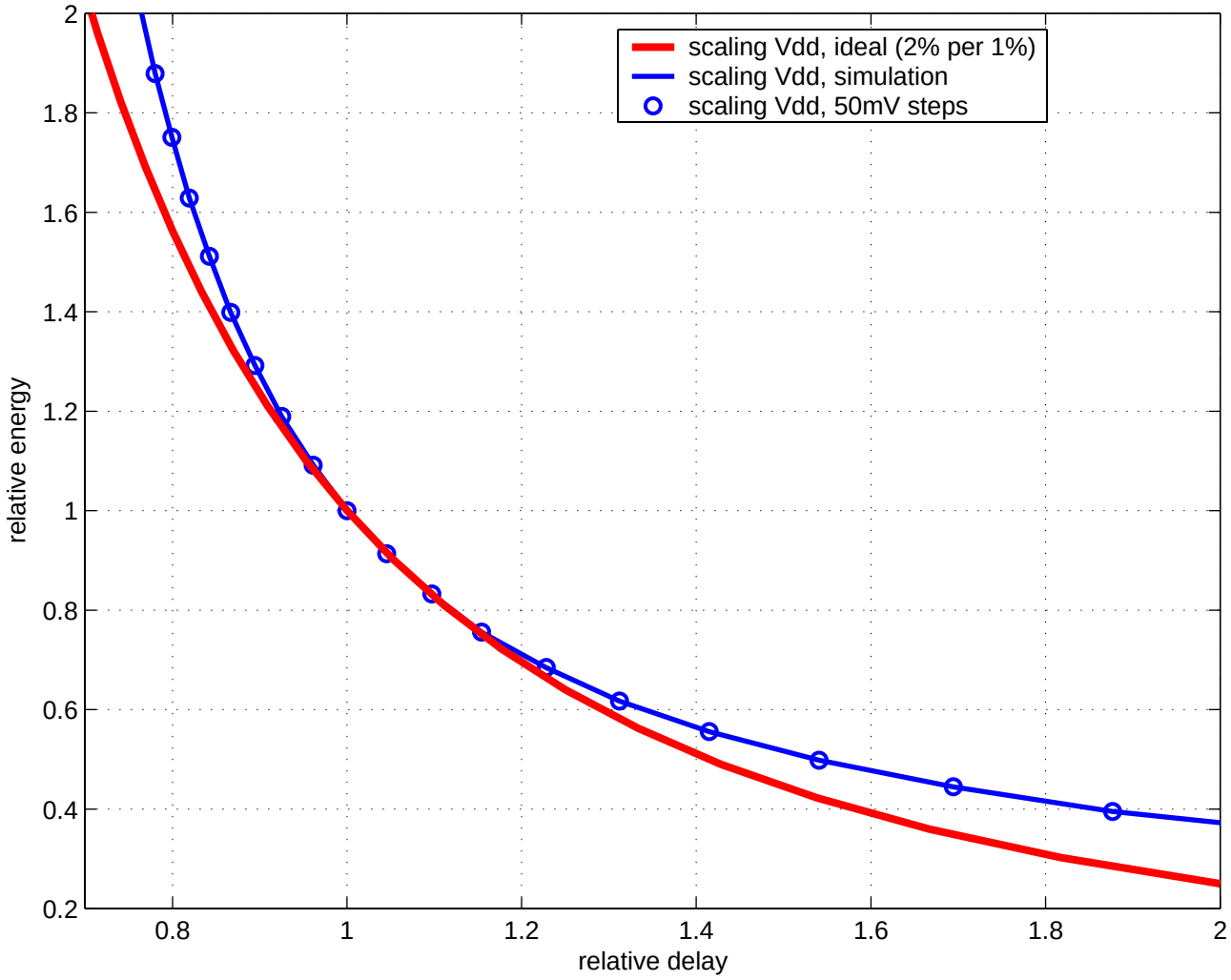
Performance

Scaling Power Supply

- Scaling *Vdd* has a known cost ("normally" 2% switch. energy for 1% performance)
  - may not be so in leakage-dominated designs
- *Vdd* can be scaled after processor is manufactured
- Voltage intensity $\Theta = \left. \frac{\%E}{\%Perf} \right|_{\text{scaling vdd}}$
- We normally tend to think that $\Theta = 2$, because $E_{\text{dynamic}} \sim V^2$, $Perf \sim f \sim V$
- In fact Θ can be from 0.5 to 3 or even higher, depending on technology and *Vdd*
- Voltage intensity is measurable

Scaling Power Supply

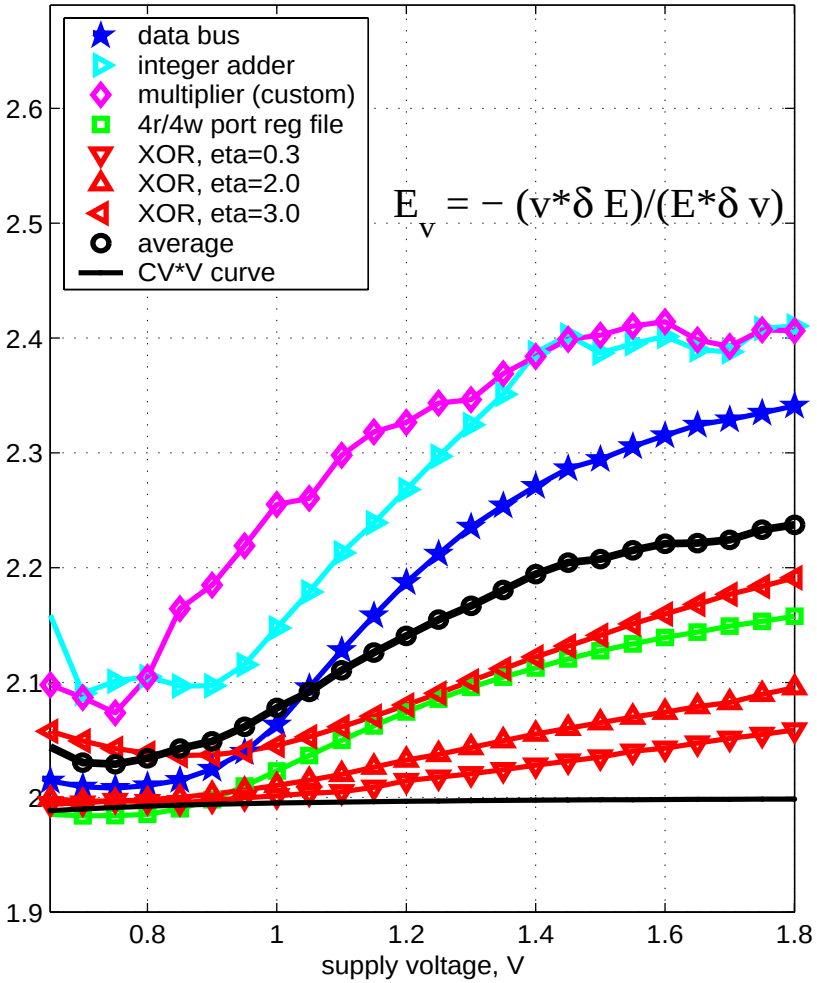
Ideal curve 2% energy per 1% delay and measured data (bulk 0.13um)



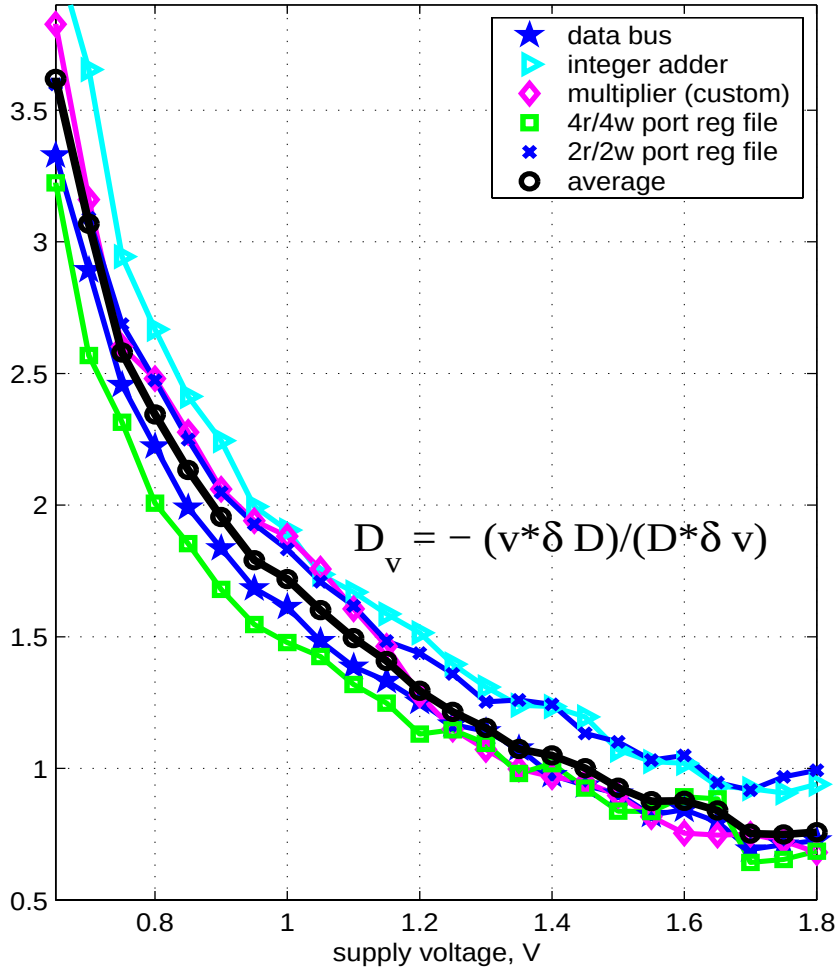
Scaling Power Supply

Measured data, 0.13um bulk

%Energy per %*Vdd*

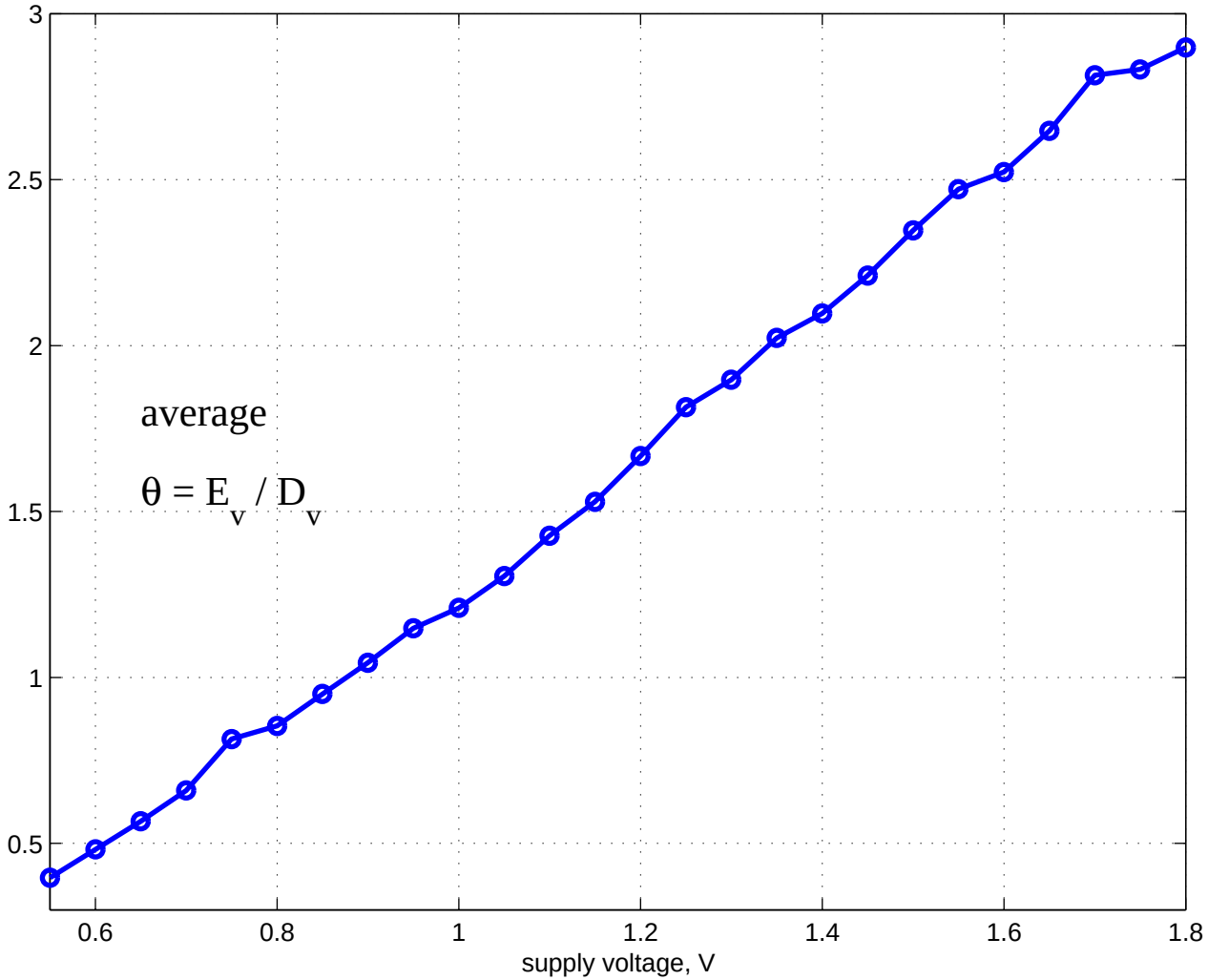


%Delay per %*Vdd*



Scaling Power Supply

Voltage Intensity (%Energy per %Delay through *Vdd* scaling), measured data, 0.13um bulk



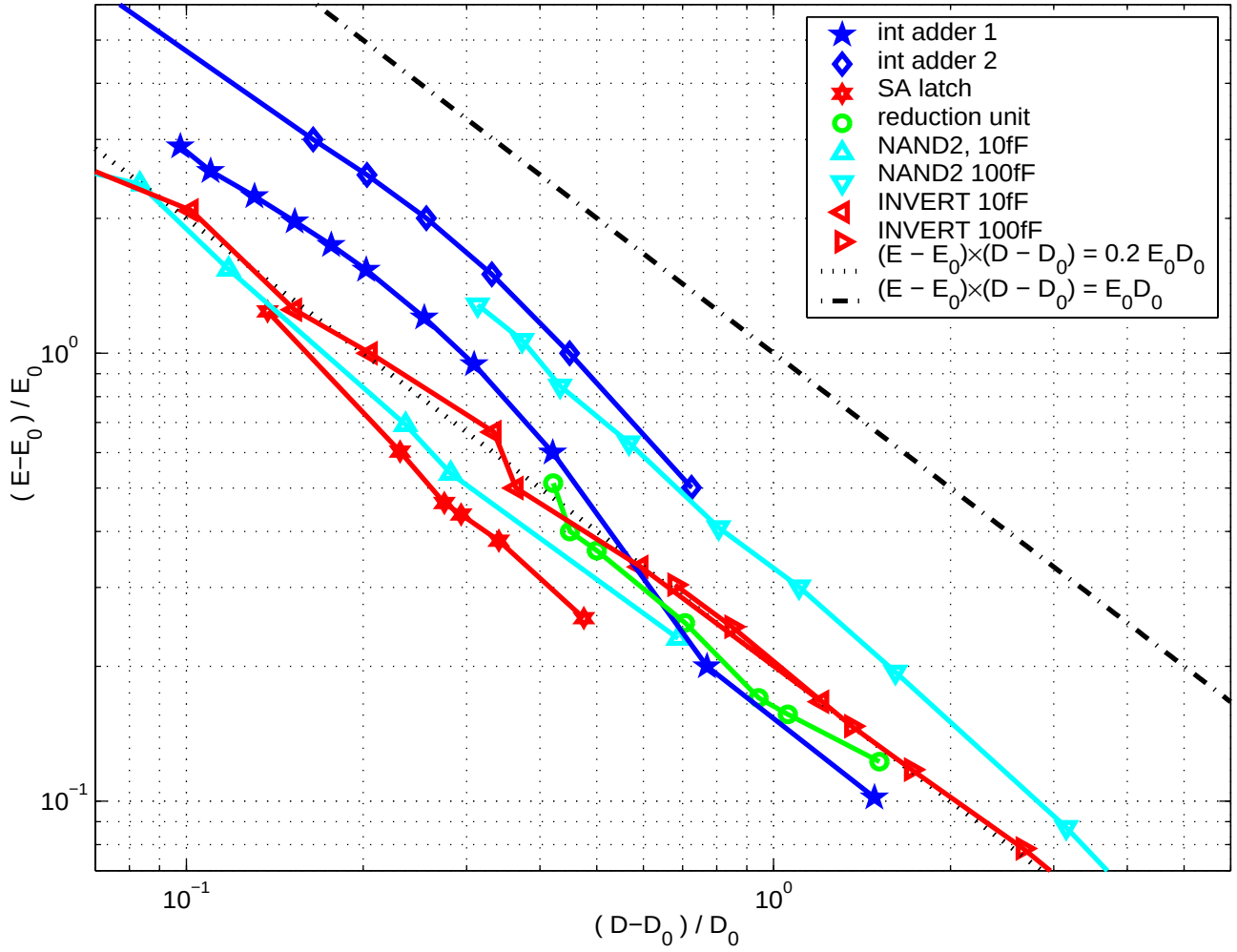
Scaling Circuits

- Second way of trading power for performance
- Involves
  - changing transistor sizes (tuning)
  - restructuring logic (performing more computations in parallel to reduce the critical path)
- As powerful as voltage scaling, but cannot be used after processor is designed
- Hardware intensity $\eta = \left.\frac{\%E}{\%Perf}\right|_{\text{scaling circuits}}$

  shows how aggressively circuits are structured and tuned to meet frequency target
- Hardware intensity can be measured
- Hardware intensity can be set as a target, using EinsTuner, $F_{cost} = \left(\frac{E}{E_0}\right)\left(\frac{D}{D_0}\right)^{\eta}$
- In “typical” designs η ranges from 0.5 to 5, but can be higher if the frequency target is too aggressive

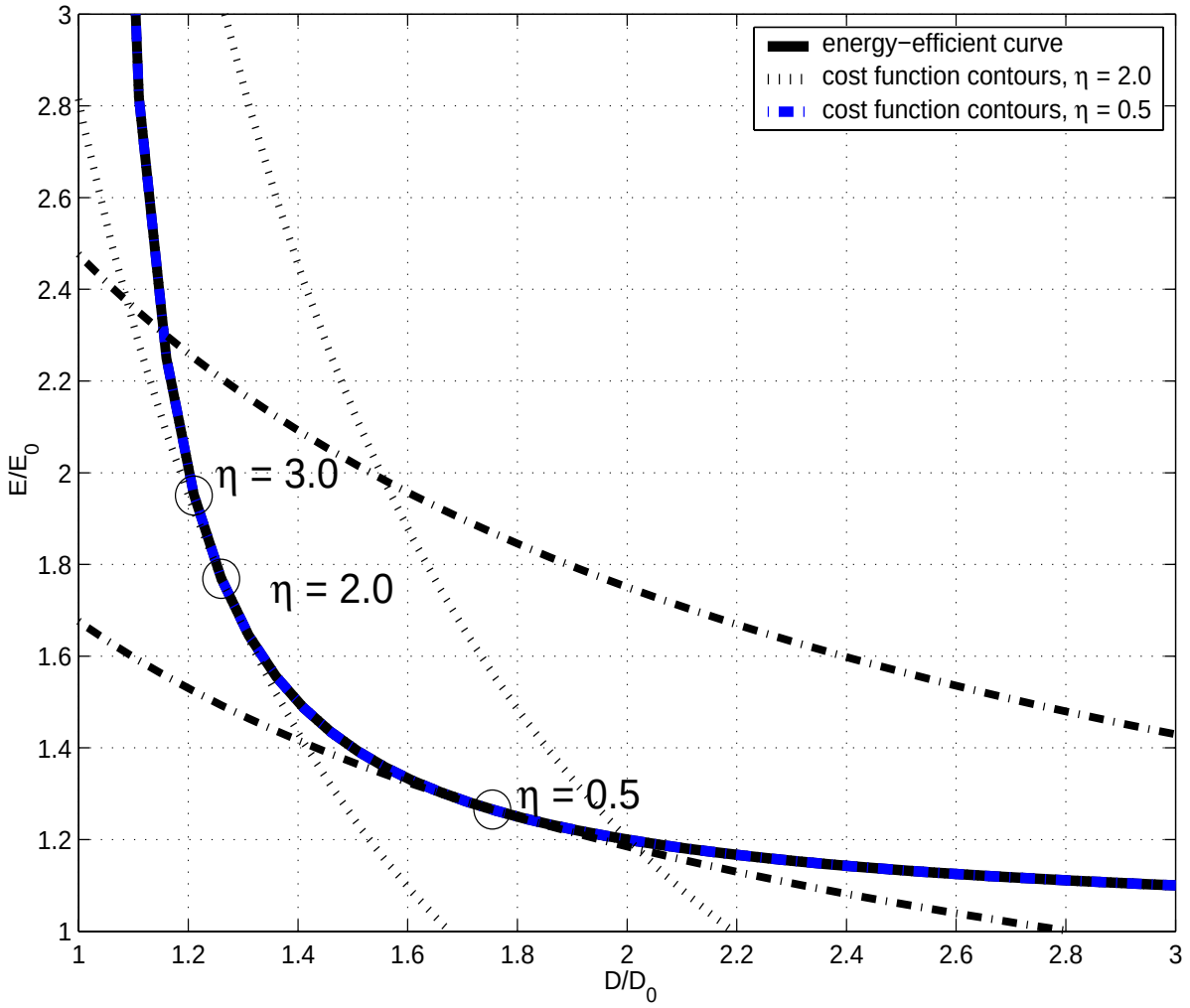
Scaling Circuits

Energy-Performance space through scaling circuits, measured data, 0.13um bulk averaged over critical circuits in eLite DSP, (J. Moreno et al. IBM JRD, vol. 47, March 2003)



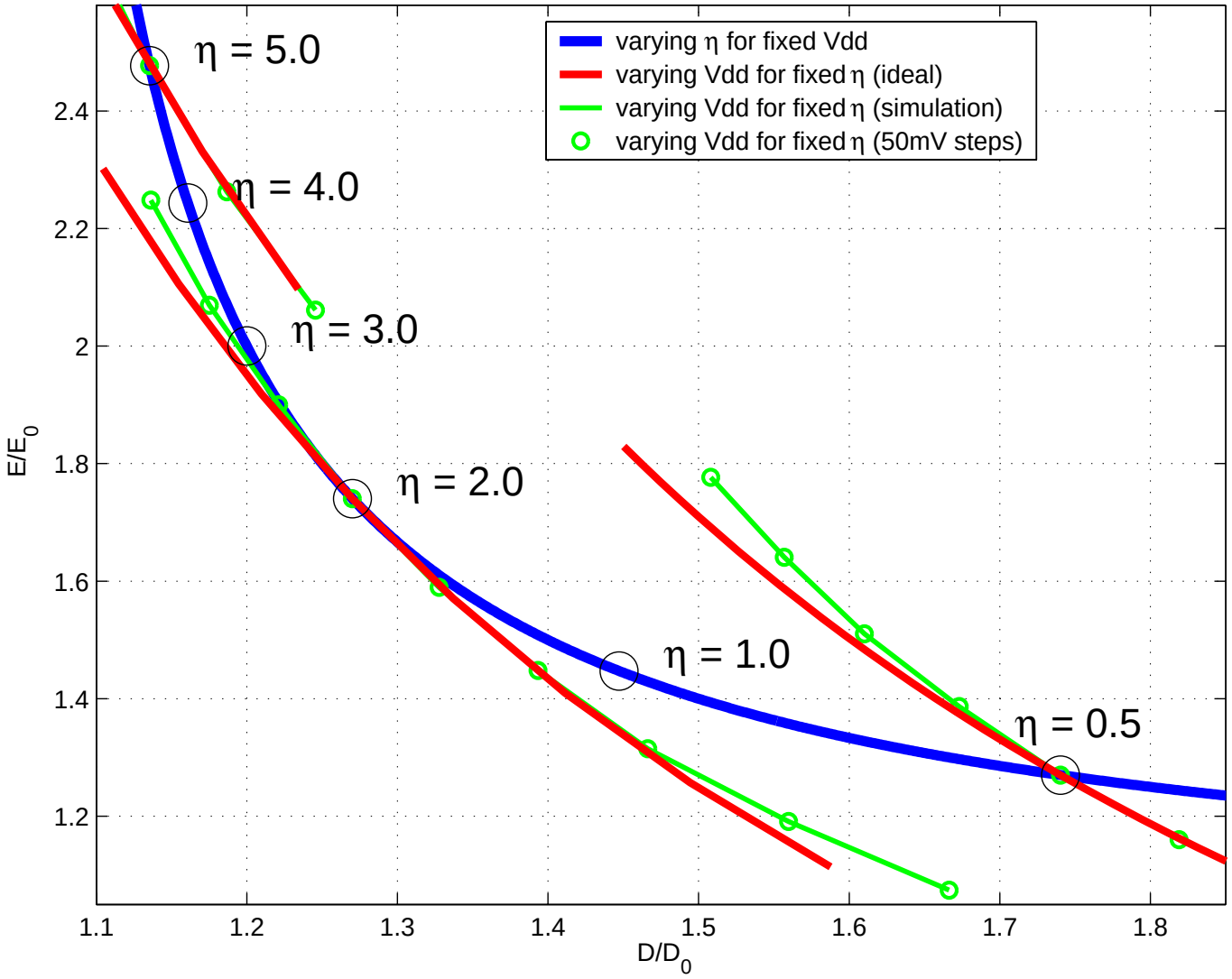
Scaling Circuits

Hardware Intensity (%Energy per %Delay through circuit scaling), typical curve

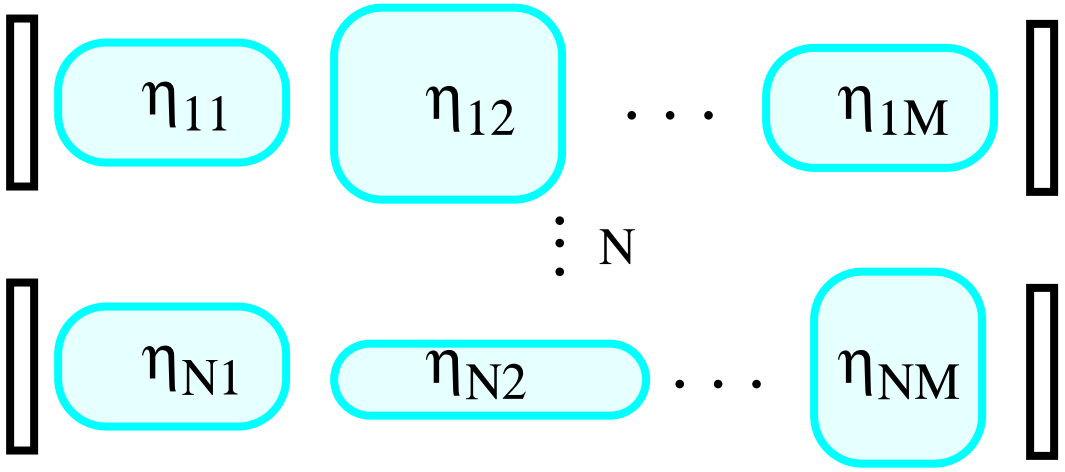


Optimal Balance, isolated macro $\eta = \theta$

For formal derivation see V. Zyuban and P. Strenski, IBM JRD, Dec. 2003, or ISLPED, 08/12/2002



Optimum Balance, a more general case



- In real pipelines different macros may be tuned for different hardware intensities
- A more general condition for the optimal balance applies, $\eta_{ag} = \Theta$
- Aggregate hardware intensity is calculated as a weighted average over all macros:

$$\eta_{ag} = \sum_{i=1}^{N} \frac{\omega_{ij}}{u_{ij}} \eta_{ij}, \text{ where } \omega_{ij} = \frac{E_{ij}}{E}, u_{ij} = \frac{D_{ij}}{T}$$

- In architectural analysis the whole processor can be represented by a single variable of aggregate hardware intensity

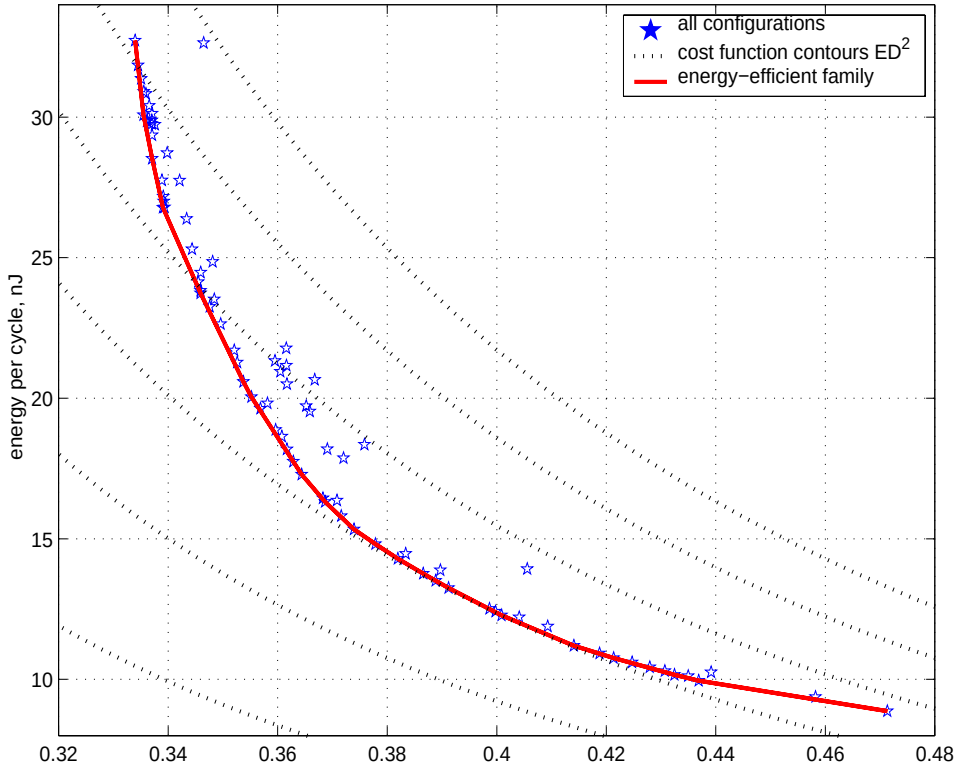
Scaling Microarchitecture

- Third way of trading power for performance
- Involves changing machine organization: pipeline depth, issue width, functions, bypasses, number of ports and entries in queues and register files, caches, branch predictors, etc.
- Even more powerful than voltage and circuit scaling, but can only be used in early stages of design
- Architectural intensity $\xi = \left. \frac{\%E}{\%Perf} \right|_{\text{scaling architecture}}$
- Architectural intensity can be measured (developed methodology)
- Architectural intensity can be set as a target

  eLite DSP example, J. Moreno et al. IBM JRD, vol. 47, March 2003

Scaling Microarchitecture

Architectural Intensity (%Energy per %Performance through scaling architecture),

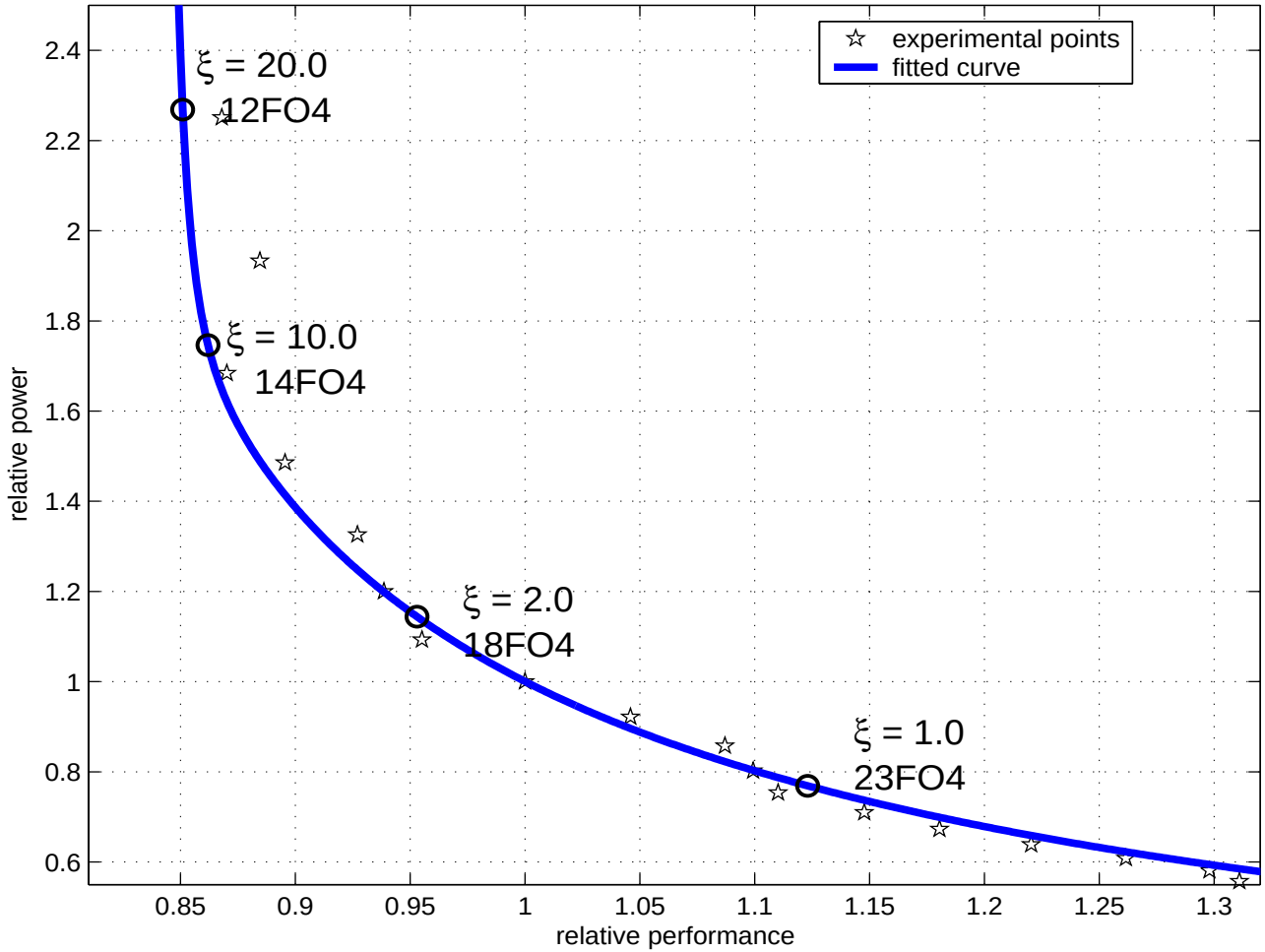


curve for an out-of-order microprocessor built by optimizing architecture for different values of γ in $\frac{\mathrm{BIPS}^{\gamma}}{\mathrm{Watt}}$, sized of 5 structures are tuned (from Ph.D. thesis V. Zyuban, 2000, also ISLPED'00)

- In designs with high ξ architectural scaling can have
  - higher power cost than *Vdd* or circuit scaling for adding performance
  - lower performance cost than frequency or circuit scaling for saving power

Example: Pipeline Depth in an OOO Processor

Srinivasan et al. MICRO-35, 11/2002



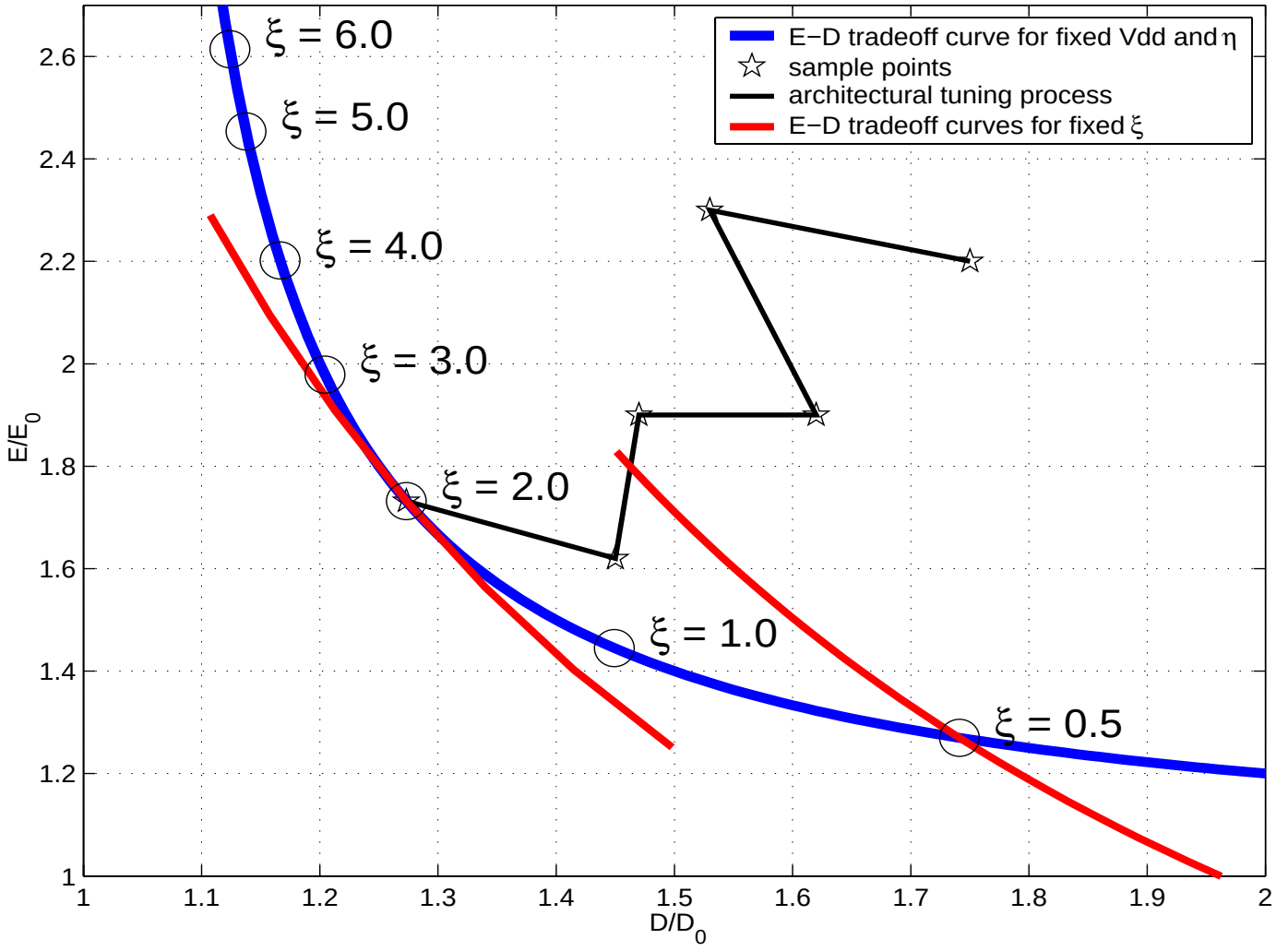
Performance

Example: Deepening Pipeline from N to $N + \Delta N$

- Assume most of the power is dissipated in latches, then neglecting other factors,
$$\left.\frac{\Delta E}{E}\right|_{\text{pipe depth}} = \frac{\Delta N}{N}$$
- Frequency in increased by $\left.\frac{\Delta f}{f}\right|_{\text{pipe depth}} = \frac{\Delta D_{\text{logic}}}{D_{\text{logic}} + D_{\text{latch}}} = \frac{\Delta N}{N}(1 - u_{\text{latch}})$,
where u_{latch} is latch insertion delay weight.
At 15FO4 $u_{latch} = 0.2$, then $\left.\frac{\Delta f}{f}\right|_{\text{pipe depth}} = 0.8\frac{\Delta N}{N}$
- Architectural speed, IPC is reduced because of longer latencies. Assume, based on architectural simulations, $\left.\frac{\Delta IPC}{IPC}\right|_{\text{pipe depth}} = -0.7\frac{\Delta N}{N}$
- Then, the net increase in performance is $\left.\frac{\Delta Perf}{Perf}\right|_{\text{pipe depth}} = \frac{\Delta f}{f} + \frac{\Delta IPC}{IPC} = 0.1\frac{\Delta N}{N}$
- At 15FO4, pipeline depth has architectural intensity $\xi = \left.\frac{\%E}{\%Perf}\right|_{\text{pipe depth}} = 10$

Optimal Balance $\xi = \eta = \theta$

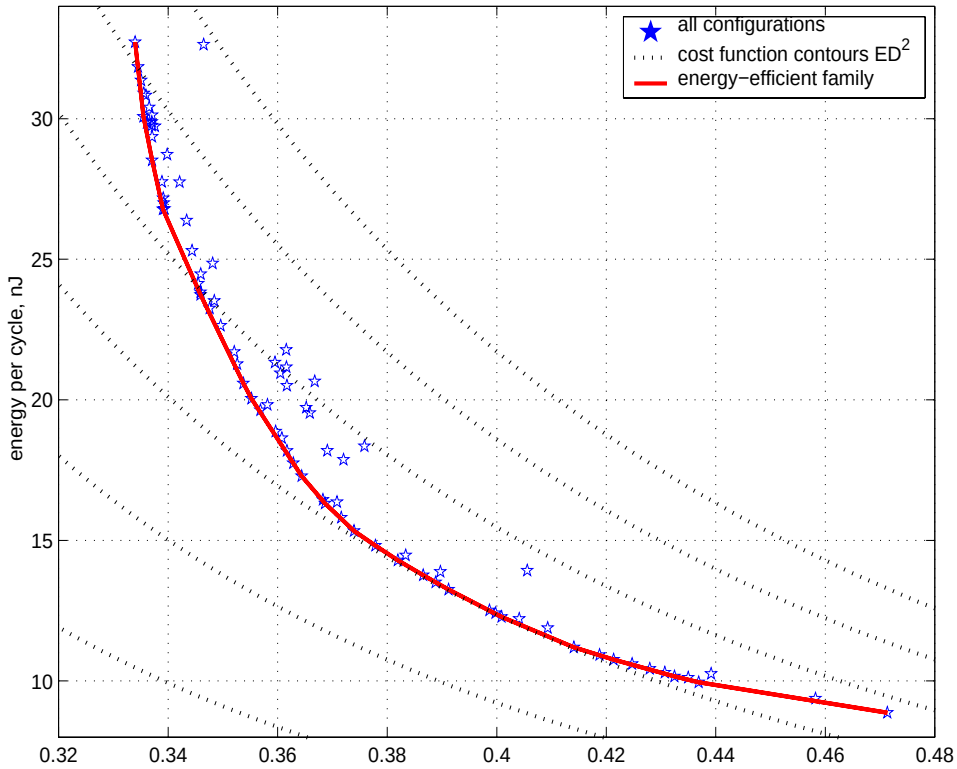
For formal derivation see V. Zyuban and P. Strenski, IBM JRD, Dec. 2003, or ISLPED, 08/12/2002



Performance

Optimum Balance $\xi = \eta = \theta$ and BIPS$^{\gamma}$ / Watt

The point at which the architectural energy-delay curve tangents a contour of cost function $F_{\text{cost}} = E \times D^{n}$ is the point that minimizes F_{cost}.



For points on the energy-delay curve

$$\xi = \left.\frac{-\Delta E / E}{\Delta D / D}\right|_{\text{fixed } v,\, \eta}, \text{ therefore}$$

$$\left.\frac{\partial E}{\partial D}\right|_{\text{fixed } v,\, \eta} = -\xi \frac{E}{D}.$$

For points on the contour of F_{cost}

$$\left.\frac{\partial E}{\partial D}\right|_{\text{fixed } v,\, \eta} = -\frac{\partial F_{\text{cost}}}{\partial D} \Big/ \frac{\partial F_{\text{cost}}}{\partial E} = -n\frac{E}{D}, \text{ therefore } n = \xi. \text{ Notice } F_{\text{cost}}^{-1} = \frac{\text{BIPS}^{n+1}}{\text{Watt}}.$$

Balance condition $\xi = \eta = \Theta$ means that $\dfrac{\text{BIPS}^{\eta+1}}{\text{Watt}}$ is the right metric.

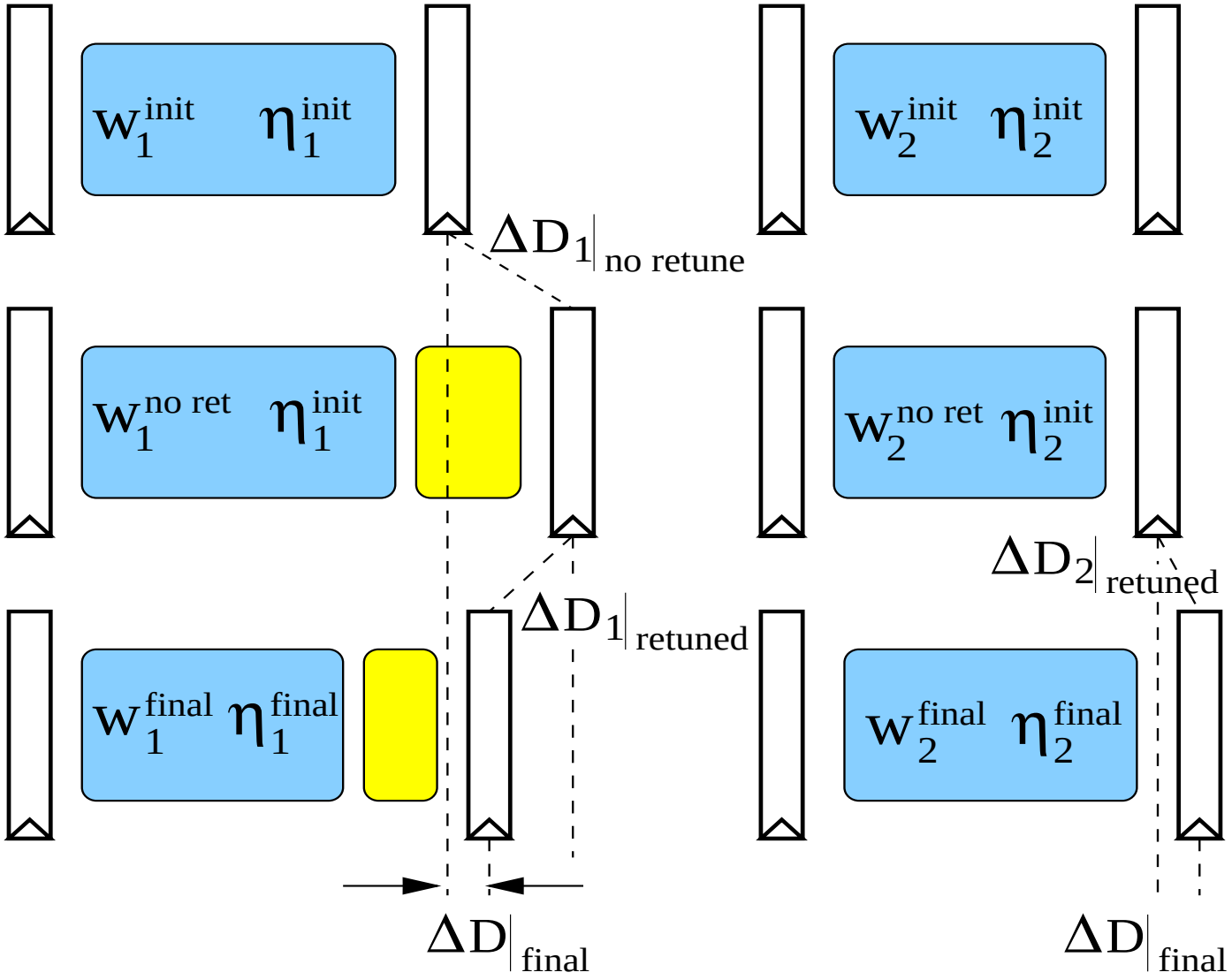
Motivation for Discrete Metric

- Optimum balance condition $\xi = \eta = \Theta$ was derived assuming the existence of a smooth energy-efficient architectural curve. In reality
  - energy-efficient curve consists of discrete points
  - getting an architecture on the energy-efficient curve is a challenge
  - need to extend the methodology to designs off the energy-efficient curve

- Derived formal method for calculating exponent in $\frac{\text{BIPS}^{\gamma}}{\text{Watt}}$, $\gamma = \xi = \eta = \Theta$
  - metric hides important assumptions
  - how to estimate Δf if an architectural feature adds logic in several pipeline stages
    - assume circuit designers will find a way to make $\Delta f = 0$
    - assume circuit designers can do nothing about it and $\Delta f = - \Delta D / D$
    - to measure $\Delta E_{dynamic}$ and $\Delta P_{leakage}$ pipeline needs to be retuned to restore the optimum balance

Retuning the Pipeline After an Architectural Modification

Stages where logic is added need to be tuned up (for higher hardware intensity)
If $\Delta f \neq 0$, circuits in the rest of the stages should be tuned down to save power



Discrete Formulation

- independent variables v, η and ξ (extended to the energy-delay space)
- functions:
  - Performance $$P(v, \eta, \xi) = \frac{f(v, \eta, \xi) I(\xi)}{N(\xi)}$$
  - Power (gated) $$W(v, \eta, \xi) = f(v, \eta, \xi) I(\xi) E(v, \eta, \xi)$$
    Power (non-gated) $$W(v, \eta, \xi) = f(v, \eta, \xi) E(v, \eta, \xi)$$
- optimization problem
  - A: minimize power $W(v, \eta, \xi)$ given a performance requirement $P(v, \eta, \xi) = P_o$
  - B: maximize performance $P(v, \eta, \xi)$ subject to power constraint $W(v, \eta, \xi) = W_o$

- need to evaluate energy efficiency of architectural modification $\Delta\xi$
- $\Delta\xi$ --> ΔV and $\Delta\eta$ to satisfy the constraint $P = P_o$ or $W = W_o$
- $\Delta\xi$ --> ΔN, Δf, ΔI and ΔE
- find relation between relative increments such that
  - A: $\Delta W < 0$ given a performance requirement $P = P_o$ $(\Delta P = 0)$
  - B: $\Delta P > 0$ subject to a power constraint $W = W_o$ $(\Delta W = 0)$

Results

- By formally solving the problem the following conditions are derived for the energy efficiency of an architectural feature
  for derivation see V. Zyuban and P. Strenski, IBM JRD, Dec. 2003, or ISLPED, 08/12/2002

- fine-grain clock gating $$\Theta\frac{\Delta I}{I} > \frac{\Delta E}{E} + \Sigma\eta_i w_i \frac{\Delta D_i}{T} + (\Theta + 1)\frac{\Delta N}{N}$$

- no clock gating $$(\Theta + 1)\frac{\Delta I}{I} > \frac{\Delta E}{E} + \Sigma\eta_i w_i \frac{\Delta D_i}{T} + (\Theta + 1)\frac{\Delta N}{N}$$

- *Vdd* - constrained ($\eta > \theta$) $$\eta\frac{\Delta I}{I} > \frac{\Delta E}{E} + \Sigma\eta_i w_i \frac{\Delta D_i}{T} + (\eta + 1)\frac{\Delta N}{N}$$

- ΔI is a projected IPC improvement from the architectural feature.
- The summation is done over all pipeline stages affected by the architectural feature.
- Terms $\frac{\Delta E}{E}$ and $\frac{\Delta D_i}{T}$ are “non-retuned” or “naive” values.
- Energy weight w_i are typically available (as targets) even at early design stages.
- Hardware intensities η_i can be extracted from previous designs or set as targets.

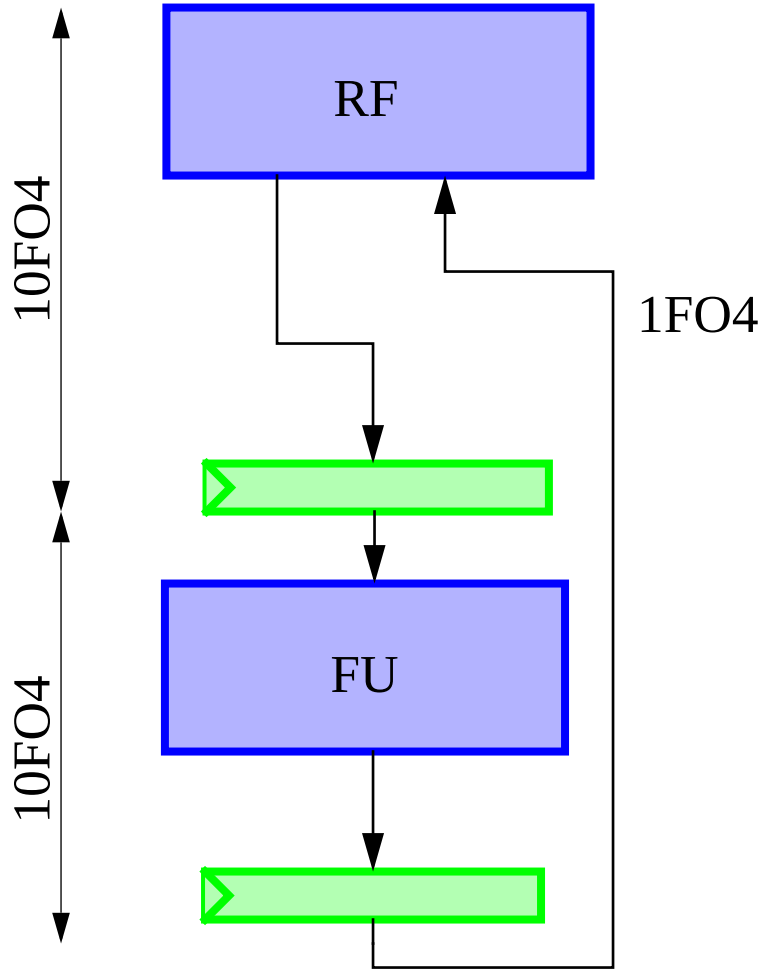
A Few Special Cases

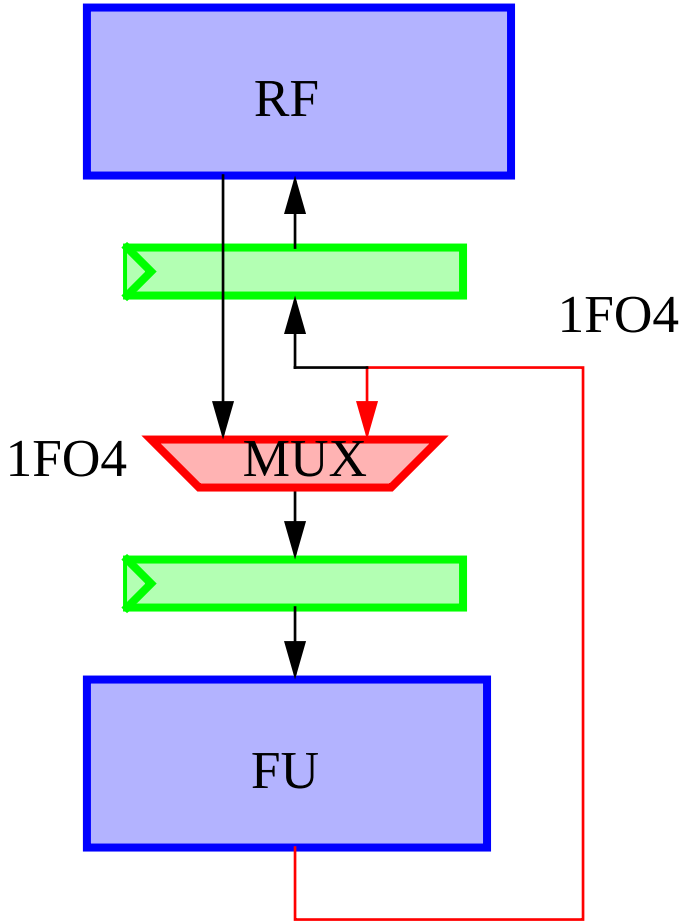
Table 1: Special Cases of the Derived Metric (0.13um, bulk)

condition	metric	equivalent prior art
$D_v >> E_v$ *(Vdd < 0.5V)* $\eta << 1$ (ultra-low power design)	$\frac{\Delta E}{E} + \frac{\Delta N}{N} < 0$	$\frac{\text{MIPS}}{\text{Watt}}$
$\theta = \eta_i = 1$ *(Vdd = 0.9V)* (eLite design point)	$\frac{\Delta I}{I} > \frac{\Delta E}{E} + \sum w_i \frac{\Delta D_i}{T} + 2\frac{\Delta N}{N}$	$\frac{\text{MIPS}^2}{\text{Watt}}$
$\theta = \eta_i = 2$ *(Vdd = 1.4V)* (nominal CU11 design point)	$2\frac{\Delta I}{I} > \frac{\Delta E}{E} + 2\sum w_i \frac{\Delta D_i}{T} + 3\frac{\Delta N}{N}$	$\frac{\text{MIPS}^3}{\text{Watt}}$
$\theta = \eta_i = 3$ *(Vdd > 1.9V)* (ultra-high performance design)	$3\frac{\Delta I}{I} > \frac{\Delta E}{E} + 3\sum w_i \frac{\Delta D_i}{T} + 4\frac{\Delta N}{N}$	$\frac{\text{MIPS}^4}{\text{Watt}}$
$\theta = 2.7\ \eta_{ag} > \theta$ *(Vdd = 1.7V)* (power supply constraint mode)	$\eta\frac{\Delta I}{I} > \frac{\Delta E}{E} + \eta\sum w_i \frac{\Delta D_i}{T} + (\eta + 1)\frac{\Delta N}{N}$	$\frac{\text{MIPS}^{\eta + 1}}{\text{Watt}}$

Example 1

A 10FO4 microprocessor A with fine-grain clock gating,
Vdd = 1.5V, θ = 2.0, 8sf technology (0.13um)
Evaluate energy-efficiency of adding execution bypass: η_{RF} = 3.0, η_{EX} = 3.0





Example 1 (cont.)

$$\left.\frac{\Delta D_{EX}}{D}\right|_{\text{no retn}} = 0.2,\ \left.\frac{\Delta D_{RF}}{D}\right|_{\text{no retn}} = 0.1,\ w_{EX} = 0.06,\ w_{RF} = 0.04$$

$\frac{\Delta E_{\text{bypass}}}{E_{\text{total}}} = 0.01$ 50% of dynamic instructions use the FXU then $\frac{\Delta E}{E} = 0.005$

$$\theta\frac{\Delta I}{I} > \frac{\Delta E}{E} + \eta_{EX} w_{EX}\frac{\Delta D_{EX}}{D} + \eta_{RF} w_{RF}\frac{\Delta D_{RF}}{D}$$

To justify adding the execution bypass $\frac{\Delta I}{I} > 0.027$ (2.7%) must be demonstrated

However, if *Vdd* < 0.9V ($\theta < 1.0$) $\frac{\Delta I}{I} > 0.053$ is necessary.

Now try using $\frac{Bips^3}{Watt}$

- assume circuit designers will find a way to make $\Delta f = 0$, get $\frac{\Delta I}{I} > 0.25\%$
- assume circuit designers can do nothing about it and $\Delta f / f = -0.2$, get $\frac{\Delta I}{I} > 20\%$

Both results are incorrect.

Example 2

Processor B: high-performance dynamic issue microprocessor with no clock gating, $Vdd = 1.7\text{V}$, $\theta = 2.7$

Evaluate energy-efficiency of adding an extra read port to a multiported int. RF.

Suppose $\frac{\Delta I}{I} = 0.02$, $\left.\frac{\Delta D_{RF}}{D}\right|_{\text{no retuning}} = 0.1$, $w_{RF} = 0.1$, $\frac{\Delta E_{\text{RF}}}{E_{\text{RF}}} = 0.2$

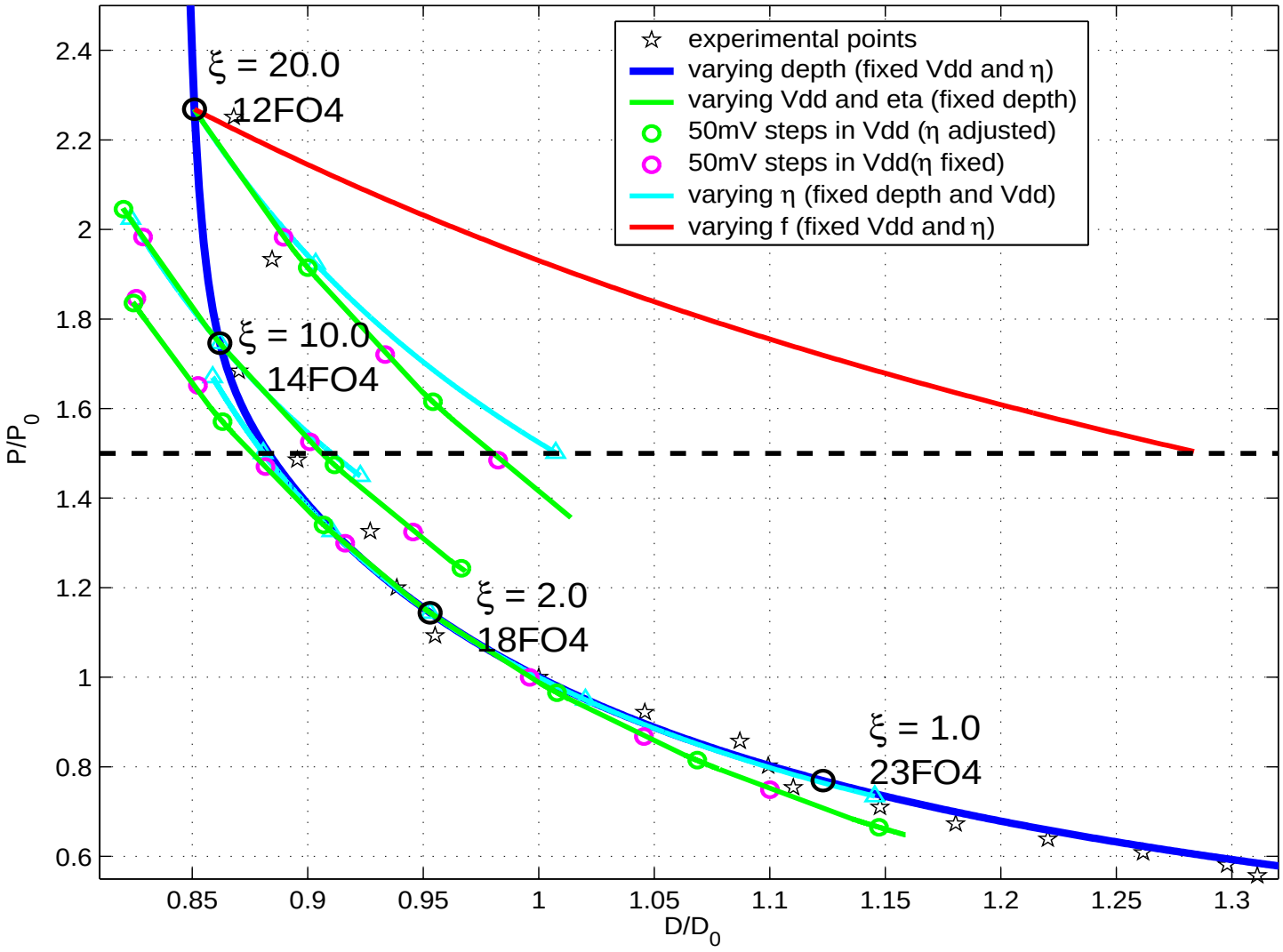
integer RF is responsible for 15% of the CPU Power, then $\frac{\Delta E}{E} = 0.03$

$$-(\theta + 1)\frac{\Delta I}{I} + \frac{\Delta E}{E} + \eta_{RF} w_{RF} \frac{\Delta D_{RF}}{D} = -3.7 \cdot 0.02 + 0.03 + 2.7 \cdot 0.1 \cdot 0.1 = -0.017 < 0$$

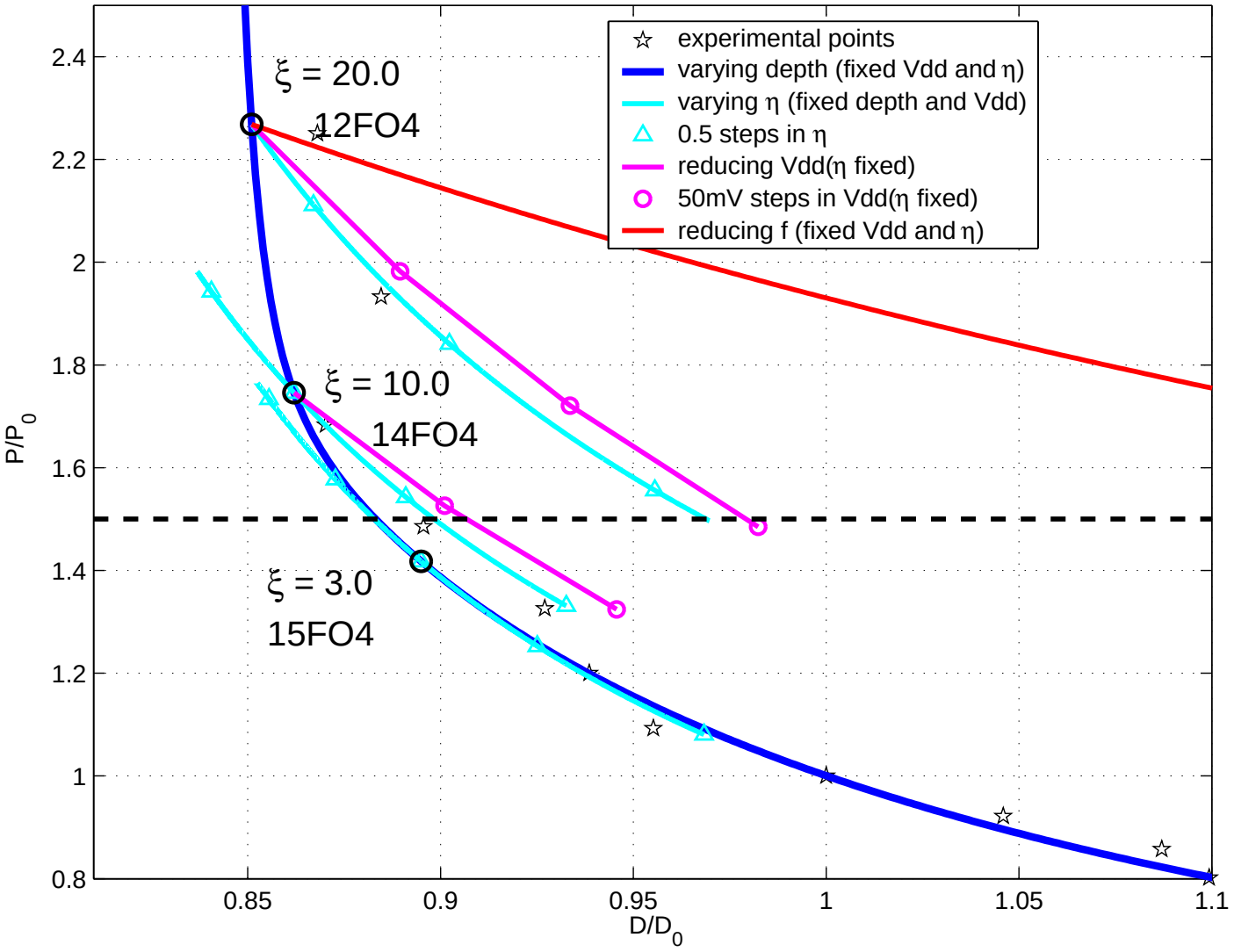
adding an extra port is energy-efficient.

However, if $Vdd < 0.9\text{V}$, the same feature is not energy-efficient.

Example 3: Optimal Pipeline Depth (4-way ooo processor)



Example 4: Optimal Pipeline Depth



End-User Power-Performance Metric

- What metric should a *customer* use to compare microprocessor products
- Customer should not base the choice on implementation details, hardware intensity of leakage currents
- It is hard to capture all requirements in a single formula (reliability, support, etc.)
- Example of how the customer-end metric can be derived
  - Yearly operating cost $Cost = C_L \cdot N + C_W \cdot Watt \cdot N$, where

    N is the number of processor cores in system

    C_L is the yearly license cost per core

    $Watt$ is the power dissipation of one core

    C_W is the energy cost per J
  - Performance of the system $Perf = Bips \cdot N$, where

    $Bips$ is the performance of a single core
  - Assume a customer needs to maximize performance without exceeding operation cost
  - Increase the number of cores ΔN (higher license cost and energy cost)
  - Use higher-performance cores $\Delta Bips$ (energy cost)

End-User Power-Performance Metric

$$\Delta Cost = C_L \cdot \Delta N + C_W \cdot Watt \cdot \Delta N + C_W \cdot N \cdot \Delta Watt = 0$$

$$\Delta Perf = Bips \cdot \Delta N + N \cdot \Delta Bips > 0$$

combine the expression, N cancels out

$\frac{\Delta Watt}{Watt} < \frac{\Delta Bips}{Bips}\left(1 + \frac{C_L}{C_E}\right)$, where

$C_E = C_W \cdot W$ is the yearly energy cost of operating one core

Under the stated assumptions the customer should choose the processor with highest

$\frac{Bips^{\gamma}}{Watt}$, where $\gamma = 1 + \frac{C_L}{C_E}$

- The end-user metric may be different from the company's energy-efficiency metric
  - There are more customers than cores
  - Companies optimize processors to maximize their profits
  - Marketing strategies traditionally emphasize performance

Conclusions

- Concept of hardware intensity is described $\eta = \left.\frac{\%E}{\%Perf}\right|_{\text{scaling circuits}}$.
- Derived conditions for an energy-efficient balance between architectural and circuit-level decisions $\xi = \eta = \Theta$.
- To achieve energy-efficient design architectural choices must be balanced with circuit-level decisions

- Different architectural decisions are optimal for different designs
- Energy-efficiency metric described $\Theta\frac{\Delta I}{I} > \frac{\Delta E}{E} + \Sigma\eta_i w_i \frac{\Delta D_i}{T} + (\Theta + 1)\frac{\Delta N}{N}$
- Relation to the $\frac{\text{BIPS}^{\gamma}}{\text{Watt}}$ metric, $\gamma = \theta + 1$ (consistent method for determining γ)

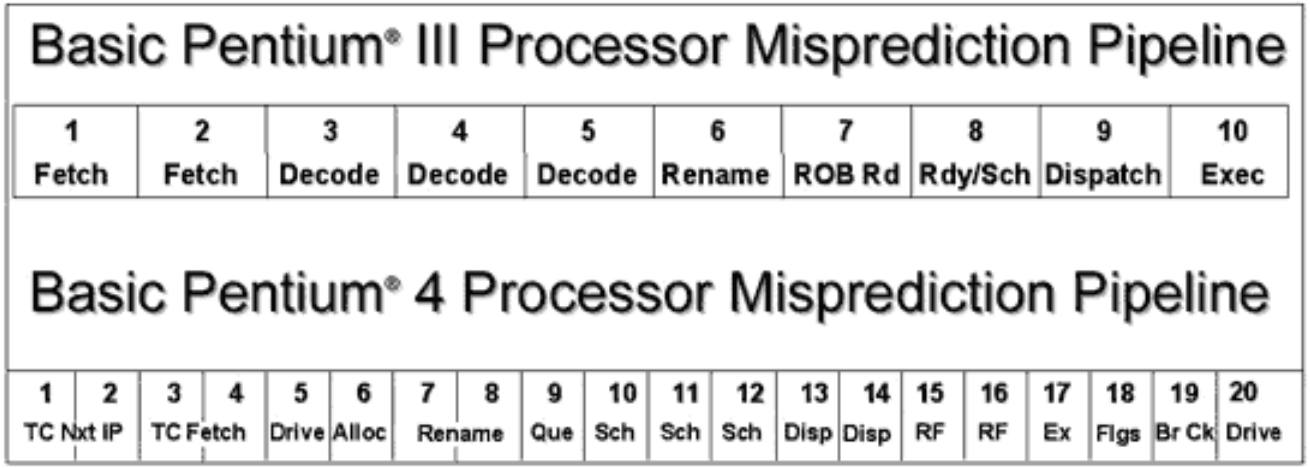
Bibliography

- V. Zyuban, P. N. Strenski, Balancing Hardware Intensity in Microprocessor Pipelines, IBM Journal of Research and Development, Volume 47, No. 5/6, pp. 585-598, 2003.
- J. H. Moreno, V. Zyuban, U. Shvadron, F. D. Neeser, J. H. Derby, M. S. Ware, K. Kailas, A. Zaks, A. Geva, S. Ben-David, S. W. Asaad, T. W. Fox, D. Littrell, M. Biberstein, D. Naishlos, and H. Hunter, An innovative low-power high-performance programmable signal processor for digital communications, IBM Journal of Research and Development, Vol. 47, No. 2/3, pp. 299-326, March/May 2003.
- V. Zyuban, P. N. Strenski, Unified Methodology for Resolving Power-Performance Tradeoffs at the Microarchitectural and Circuit Levels, Proceedings of IEEE Symposium on Low Power Electronics and Design, pages 166-171, August 2002.

Tutorial Outline

	Introduction and Motivation
	Basics of Performance Modeling
	- Turandot performance simulation infrastructure
	Architectural Power Modeling and Metrics
	- PowerTimer extensions to Turandot
	- Power-Performance Efficiency Metrics
10:30-11:00	Case Studies and Examples
	- Optimal Power-Performance Pipeline Depth
	Validation and Calibration Efforts
	Future challenges and Discussion
	Bibliography

Intel Pipeline Depths



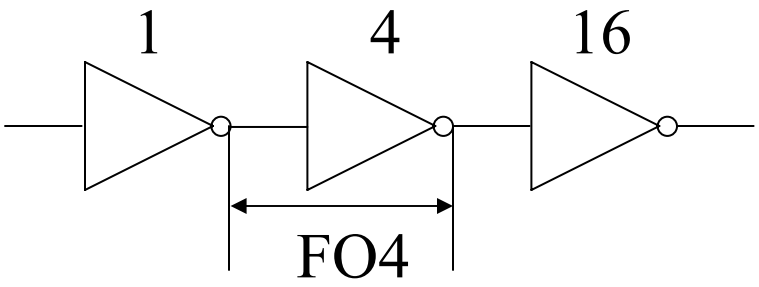
- Pipeline Depth is key to microprocessor performance
- Pentium III: 10 pipestages
- Pentium 4: 20 pipestages
- Intel @ ISCA2002: 52 pipestages is optimal [Sprangle02]

Overall Methodology

- Begin with a base, core microarchitecture
- Develop energy models based on detailed circuit-level power analysis of macros from an existing machine
- Develop energy scaling equations for pipeline depth
- Study the sensitivity of the energy model parameters to the optimal pipeline depth

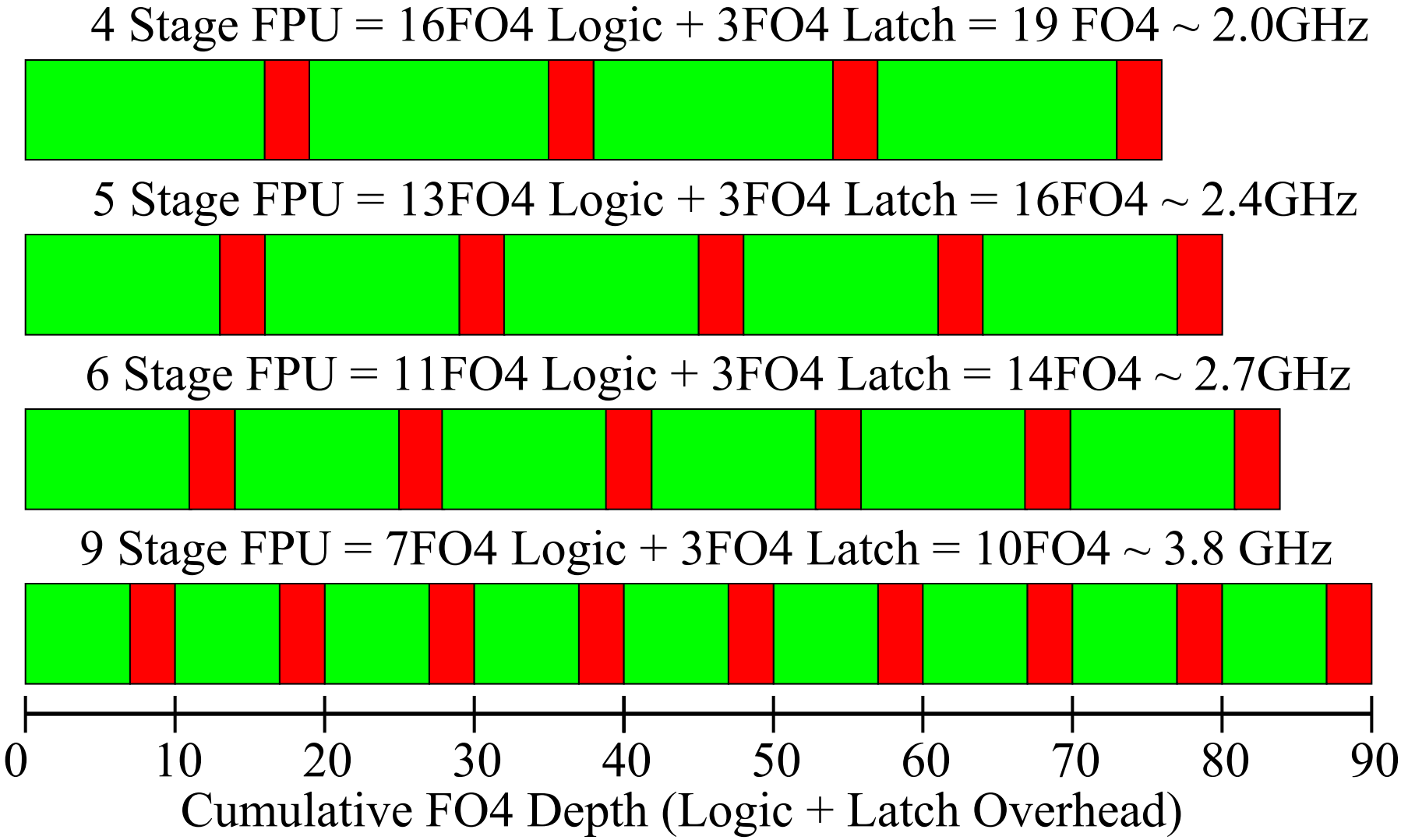
Background/Definitions

- Fanout-of-4 inverter metric (Horowitz)

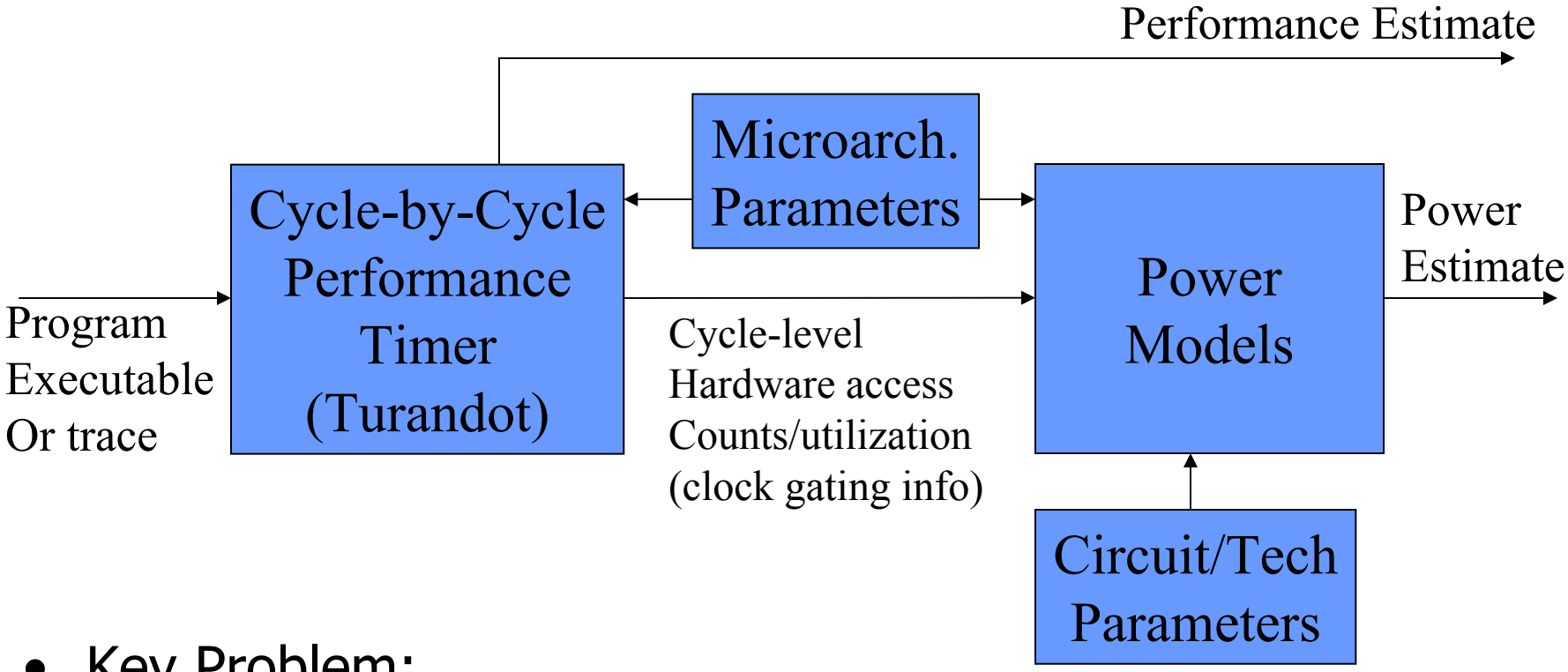


- Delay of an inverter with $C_{load}/C_{in}=4$
- More or less stable for process, voltage, temperature
- We use this to measure amount of logic per stage of the pipeline

Pipeline Scaling Methodology



PowerTimer



- Key Problem:
  - How to scale energy models for pipeline *depth* rather than just pipeline *width*

Energy Model Formation

- Energy models based on circuit-level power analysis of structures in current high-performance PowerPC processor
- Power analysis
  - For each macro collect *ungated* power (ckt sim)
    - Clocking power (latches, LCBs, array clocking)
    - Active power (Logic, data-dependent array)
    - Leakage power
  - Clock gating factors determined based on utilization and macro-level clock gating eligibility

Factors Affecting Choice of Pipeline Depth

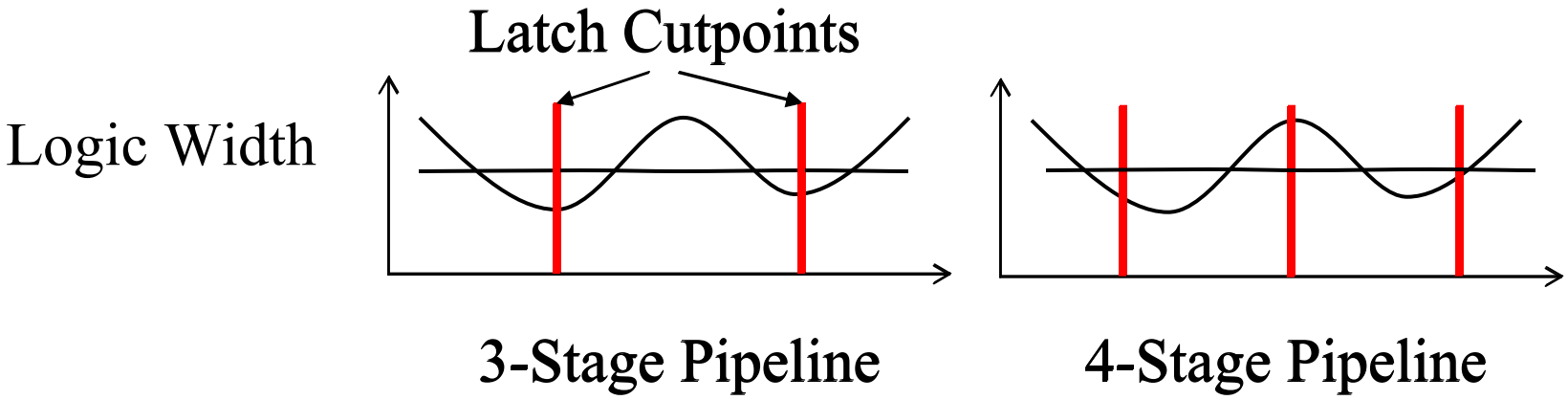
- Cycles-Per-Instruction (CPI)
- Clock Frequency
- Clock Gating Effects
- Latch-to-Logic Dynamic Power Ratio
- Latch Growth Factor
- Glitching Activity
- Leakage Power Scaling
- Power-Delay Ratios for Latches and Logic

Energy Model Scaling: CPI, Frequency, Clock Gating

- CPI impacts performance only (workload dependent)
- Clock Gating impacts power only (workload dependent)
- Frequency impacts both

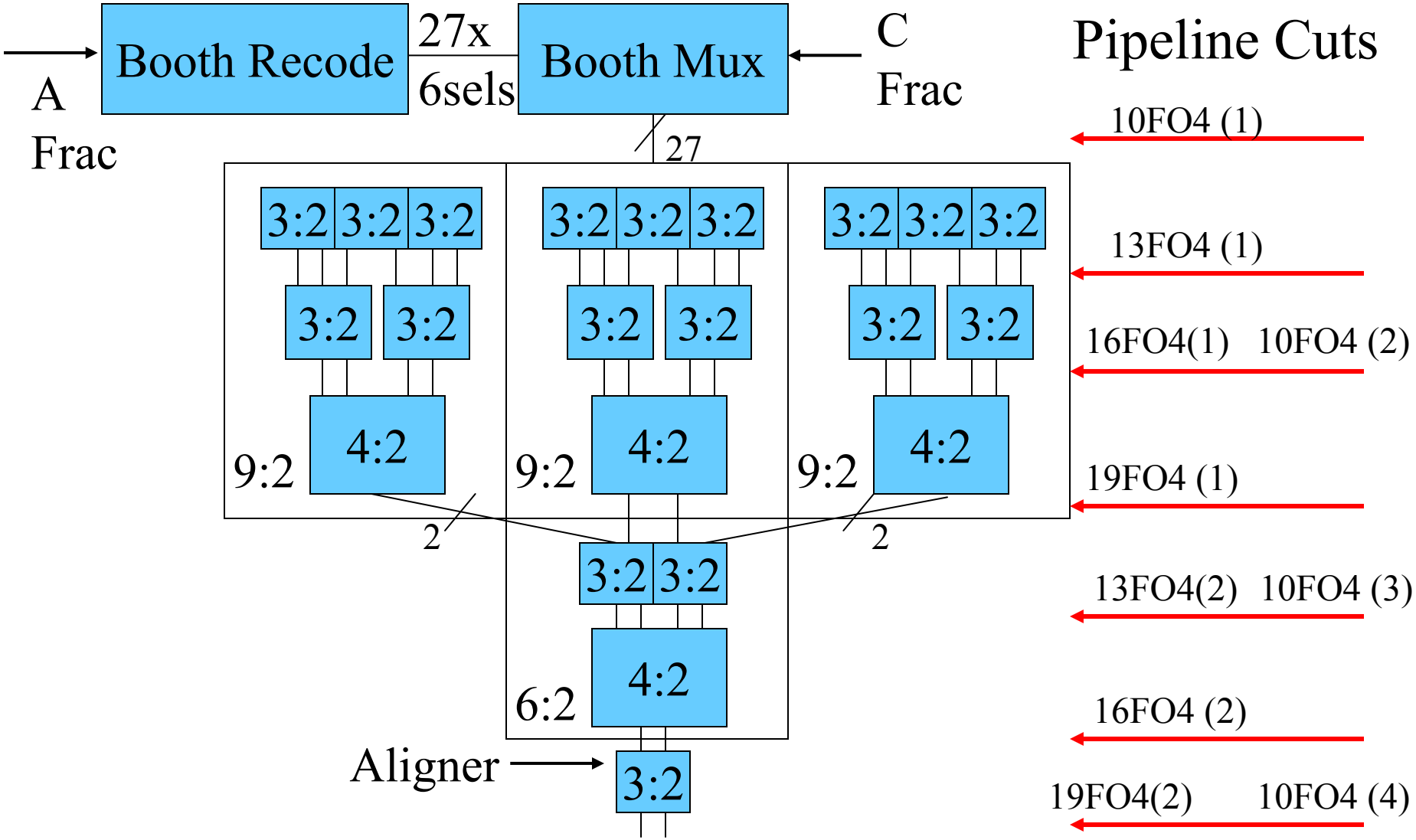
Energy Model Scaling: Latch Growth Factor, Latch-Logic Ratio

- Latch growth has a big impact
  - Logic shape functions are often not flat

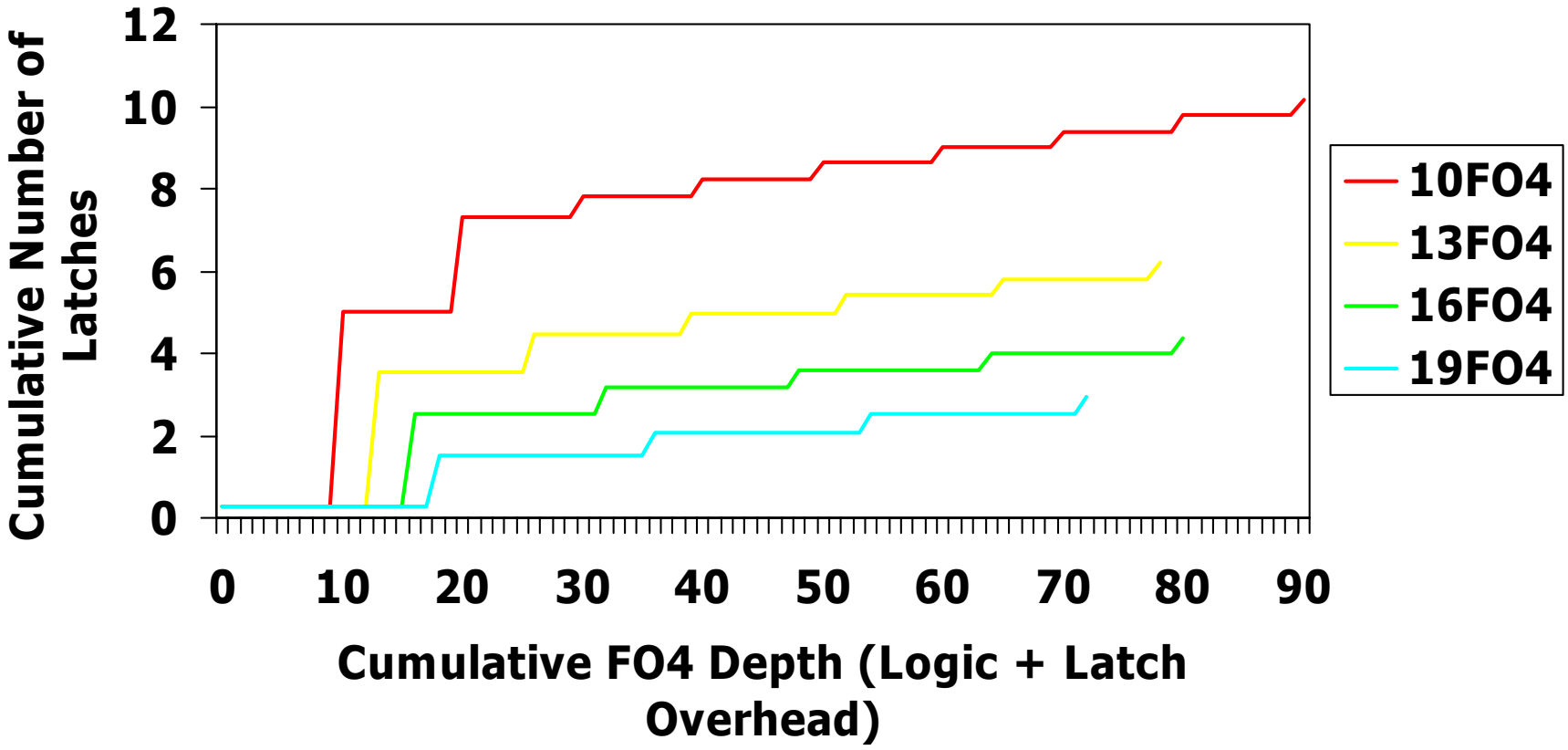


- LatchScale = (Latch-logic power ratio) * (base FO4/FO4)^{LGF}
- Latch Growth Factor slightly super-linear (1.1)
- Latch-Logic Power Ratio of current machines (70%-30%)

Energy Model Scaling: Latch Growth Factor



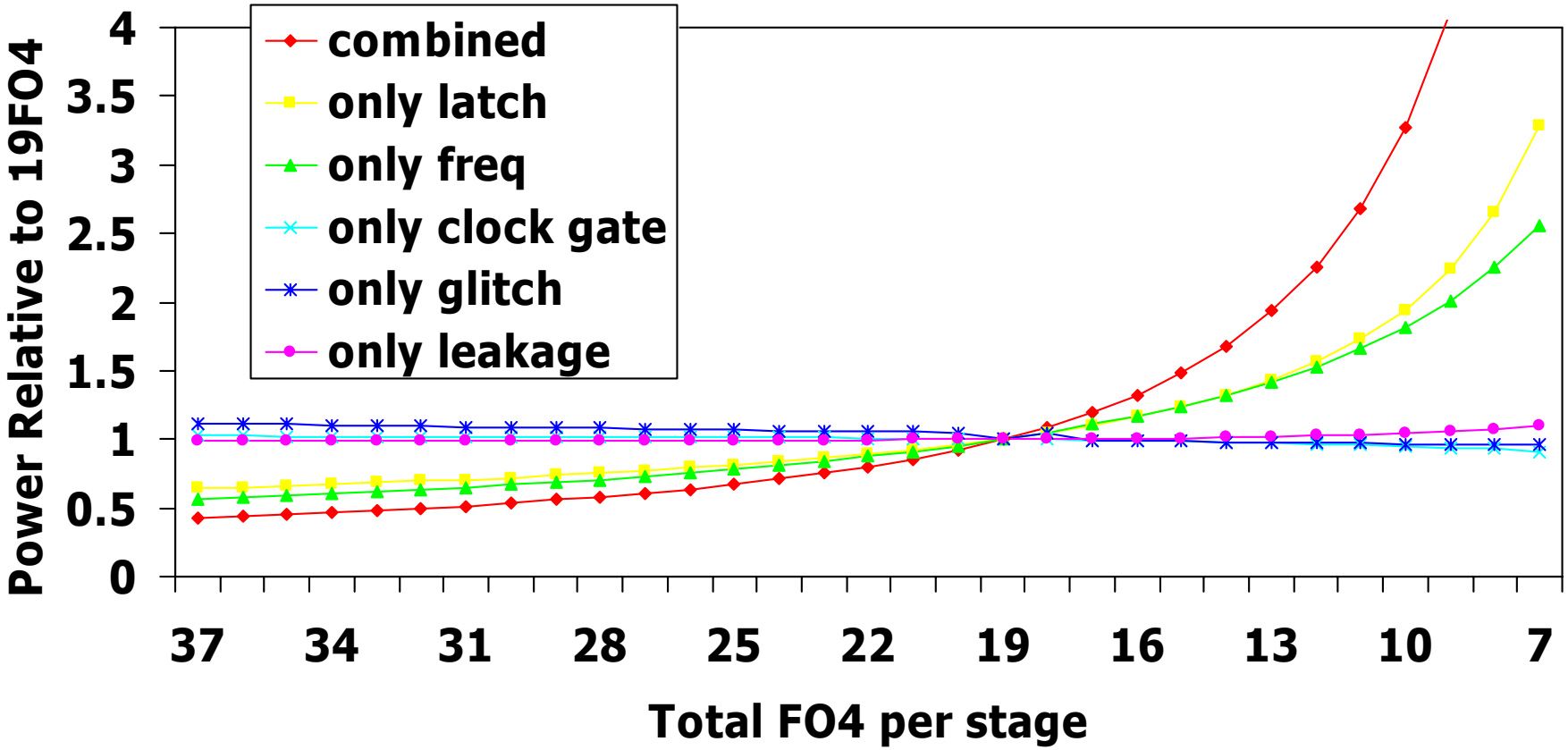
Energy Model Scaling: Latch Growth Factor



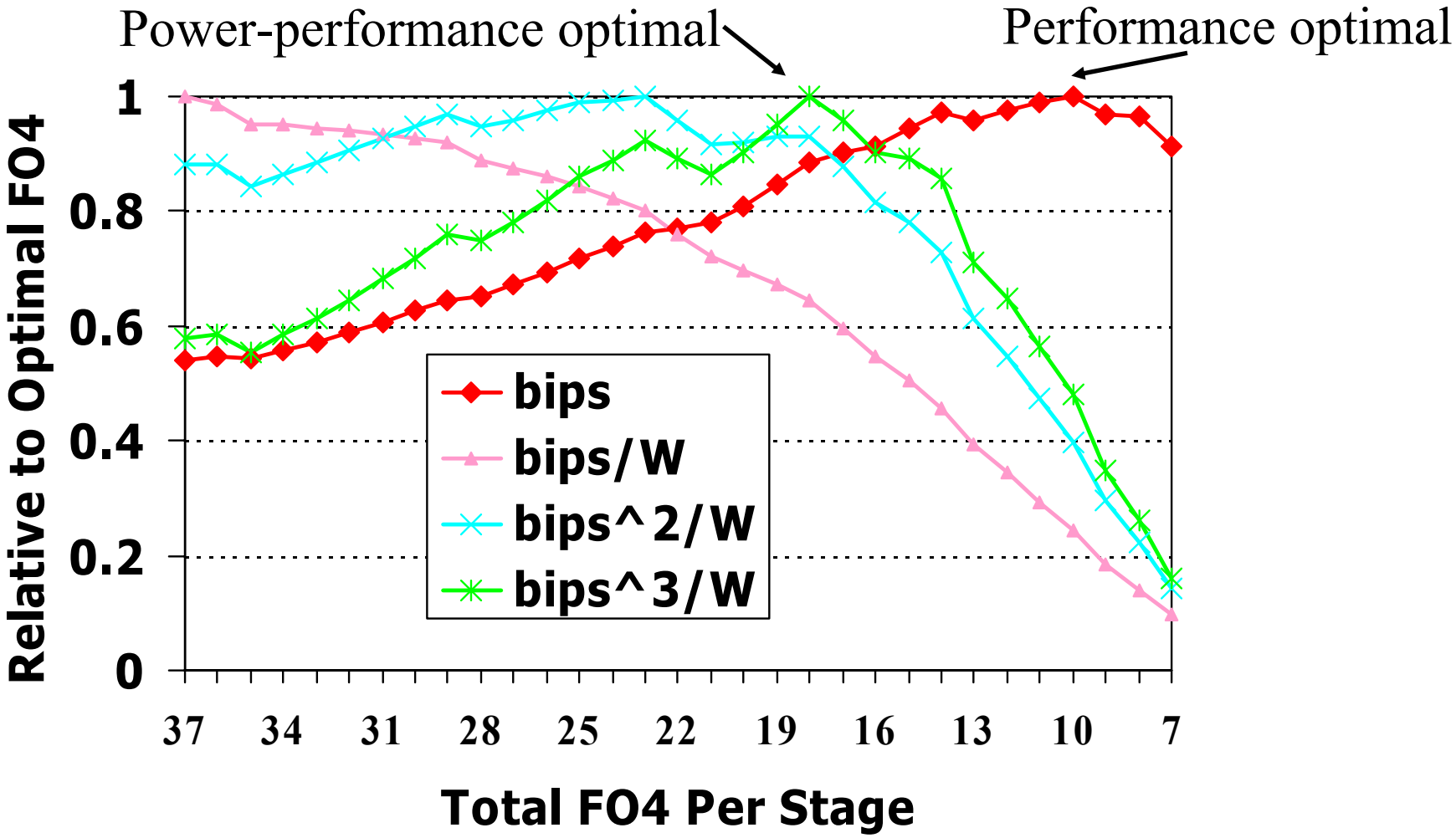
Energy Model Scaling: Glitching and Leakage

- Glitching *reduces* with deeper pipelines
  - More pipeline latches stop glitch propagation
- Leakage power component grows more slowly than dynamic power component with deeper pipelines
  - Leakage does not scale with frequency
  - Leakage growth is proportional to overall width of latches rather than overall power of latches
    - Overall Latch width % << Overall Latch power %

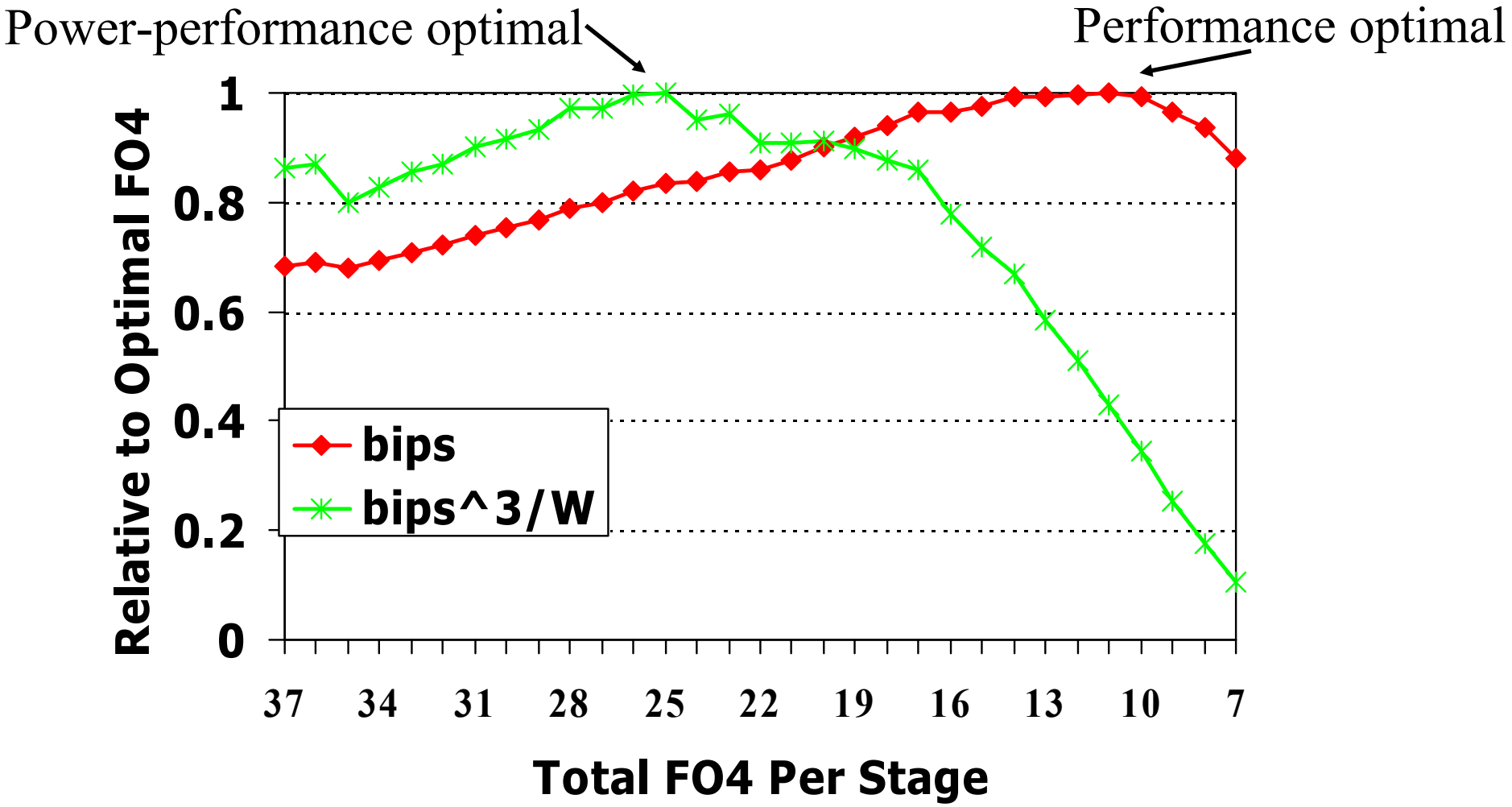
Power Scaling Effects



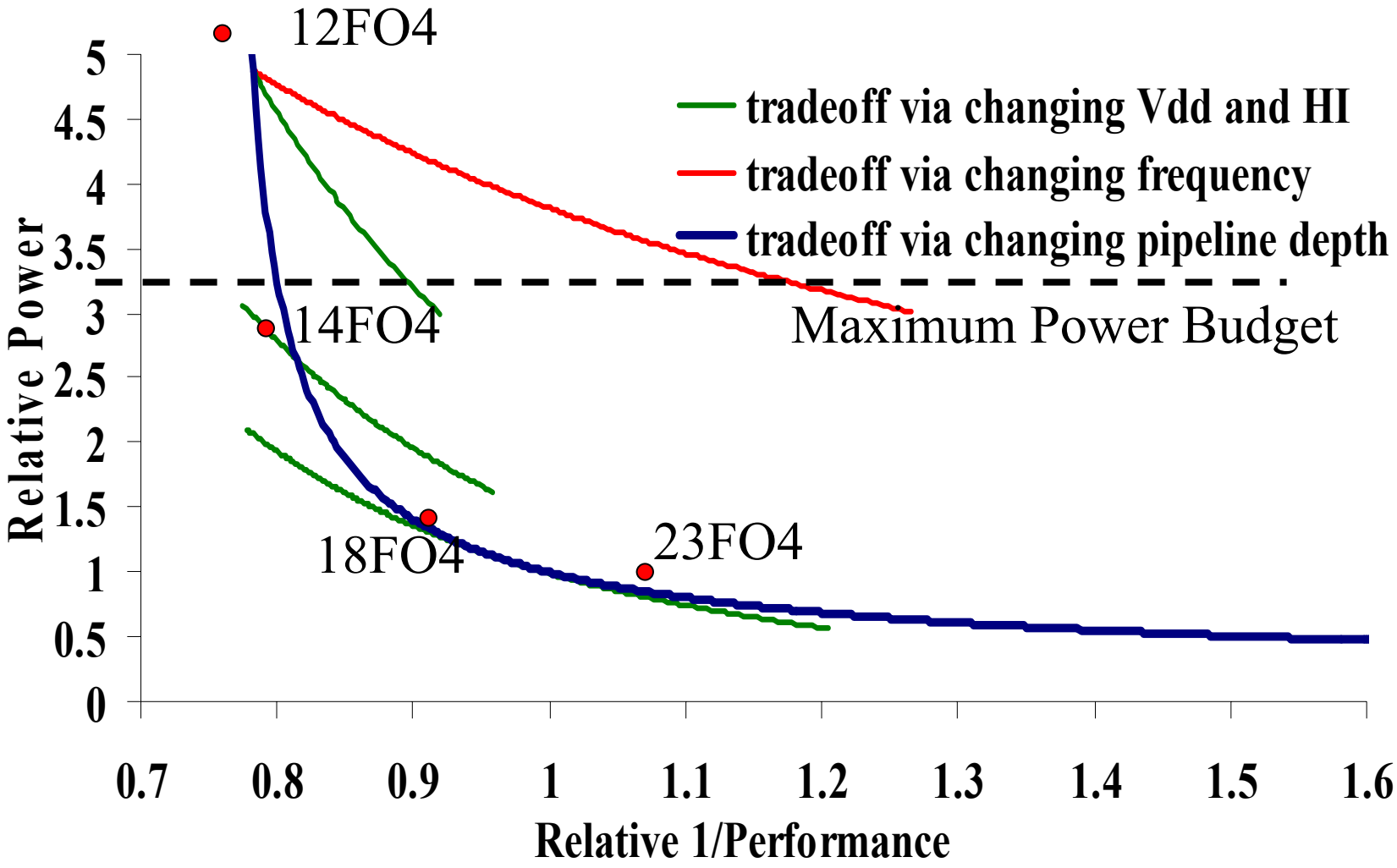
Scaling Results: Average of SPEC2K



Workload impact: TPCC Trace



Impact on Design



Temperature/Power Density Analysis

Temperature "landscape": space and time
How to estimate early in the design cycle?

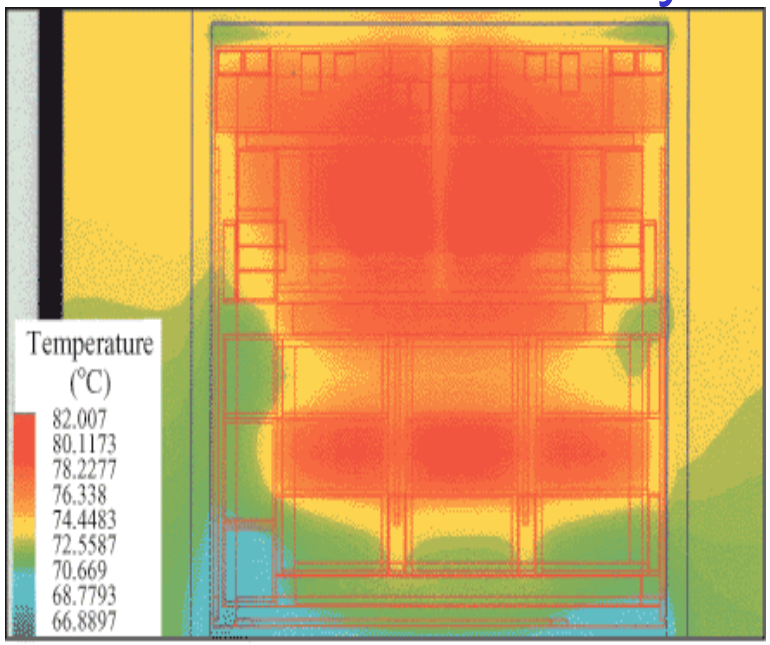


Figure 19

Map of FET junction temperatures for a 115-W packaged POWER4 chip derived from the chip power analysis and thermal modeling simulations described in the section on distribution.

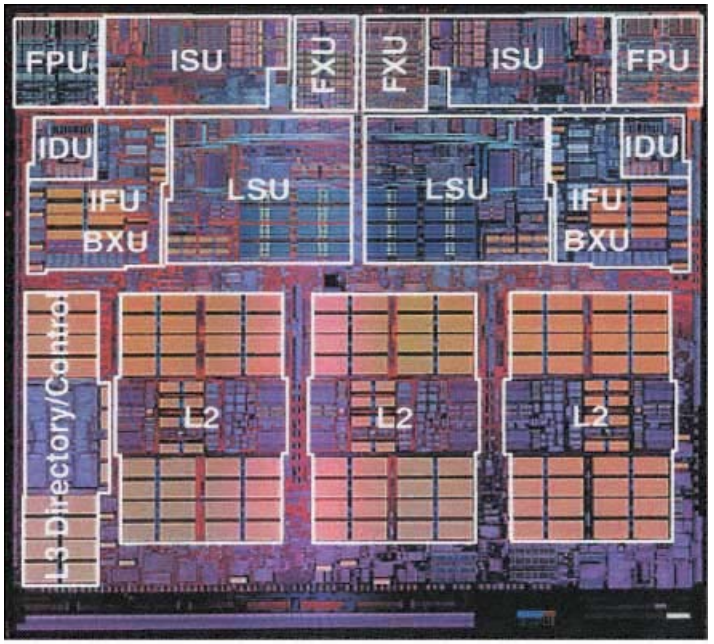


Figure 2

POWER4 chip photograph showing the principal functional units in the microprocessor core and in the memory subsystem.

From IBM Journal of R&D, Vol. 46, No. 1, 2002

Integration with UVA's HotSpot Project

- Initial work has begun on integrating PowerTimer with HotSpot [Skadron, Stan, et. al., ISCA2003]
  - Allows early-stage temperature analysis, hotspot identification
  - Thermal-aware microarchitecture design

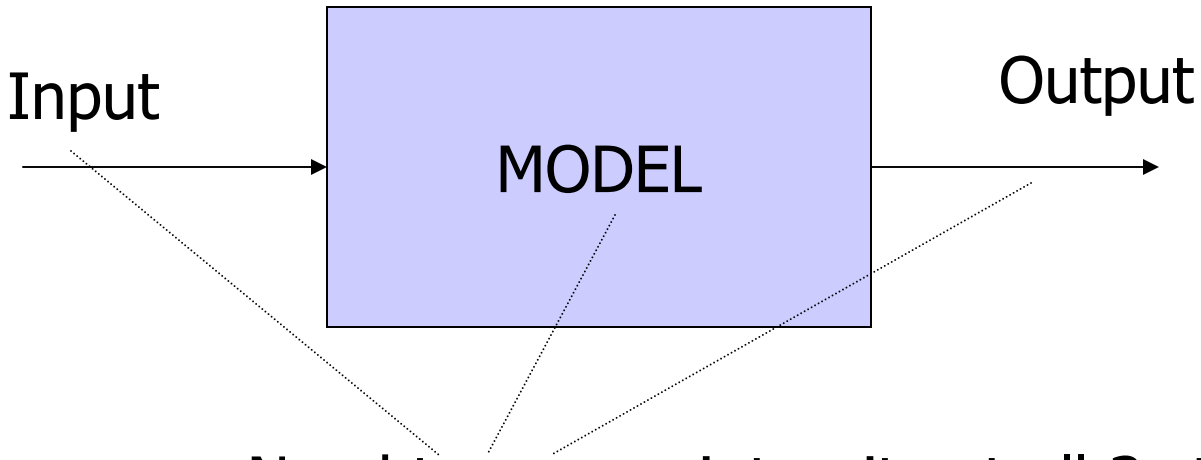
Bibliography: PowerTimer Case Studies

- David Brooks, John-David Wellman, Pradip Bose, and Margaret Martonosi. "Power-Performance Modeling and Tradeoff Analysis for a High-End Microprocessor," Workshop on Power-Aware Computer Systems (PACS2000, held in conjuction with ASPLOS-IX), Cambridge, MA., November, 2000.
- Viji Srinivasan, David Brooks, Michael Gschwind, Pradip Bose, Victor Zyuban, Philip N Strenski, and Philip G Emma, "Optimizing Pipelines for Power and Performance," 35th International Symposium on Microarchitecture (MICRO-35), November, 2002.
- David Brooks, Pradip Bose, Viji Srinivasan, Michael Gschwind, Philip G. Emma, Michael G. Rosenfield. "New methodology for early-stage, microarchitecture-level power-performance analysis of microprocessors," IBM Journal of Research and Development, Volume 47, No. 5/6, 2003.

Tutorial Outline

	Introduction and Motivation
	Basics of Performance Modeling
	- Turandot performance simulation infrastructure
	Architectural Power Modeling
	- PowerTimer extensions to Turandot
	- Power-Performance Efficiency Metrics (Victor)
	Case Studies and Examples
	- Optimal Power-Performance Pipeline Depth
11:15-11:45	Validation and Calibration Efforts
	Future challenges and Discussion
	Bibliography

Validation



Input Validation: making sure that the input, e.g. trace, is representative of the workloads of interest

Model Validation: ensuring that the model itself is accurate

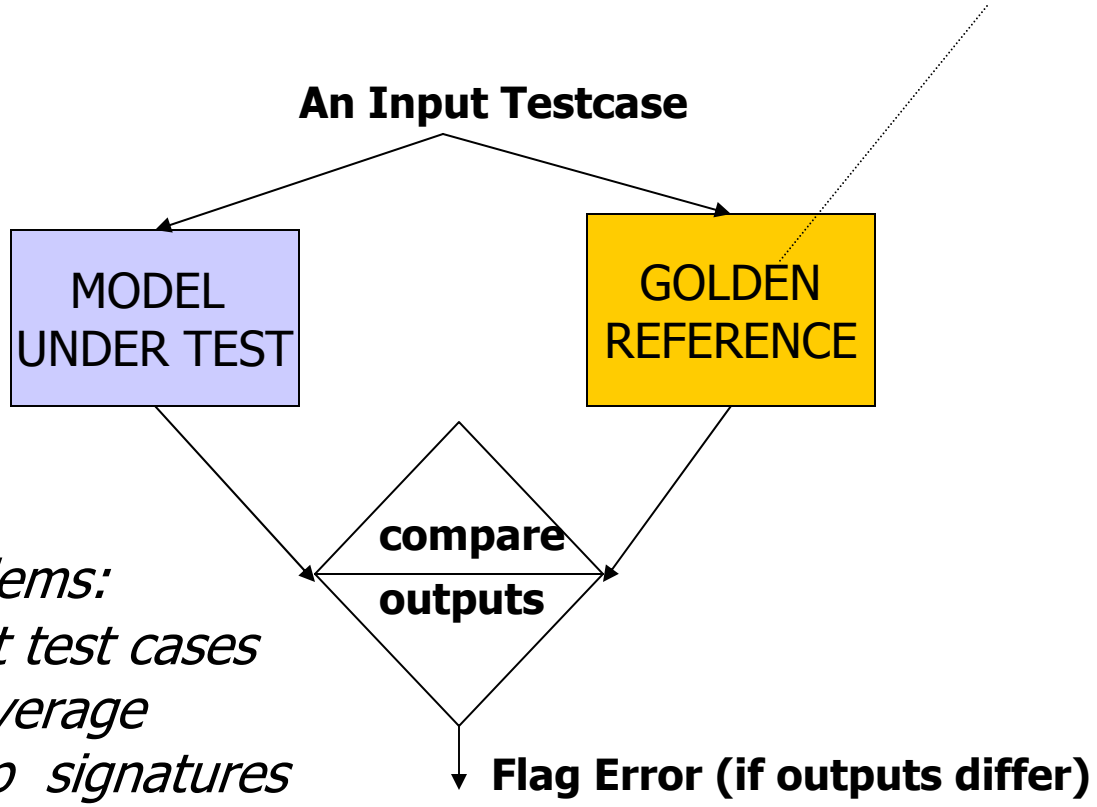
Output Validation: interpreting the results correctly

Post-Silicon Calibration Lessons Learnt from an early ’90s development project

- Trace sampling to reduce simulation time must be done with care!
  - Trace input inaccuracy was the biggest source of error
  - Later research invented R-metric to quantify inaccuracy in sampled traces (Iyengar, Trevillyan, Bose, HPCA-96)
- Statistical methods of simulating stall effects (like cache misses) are prone to large overall errors
  - This was the second largest source of error
- Neglecting the effect of kernel code in traces can be dangerous in some cases
  - Execution-driven simulation that captures kernel code is desirable
- Cycle-level cross-validation against pre-silicon, detailed reference models (RTL or pre-RTL) is definitely needed

Model Validation

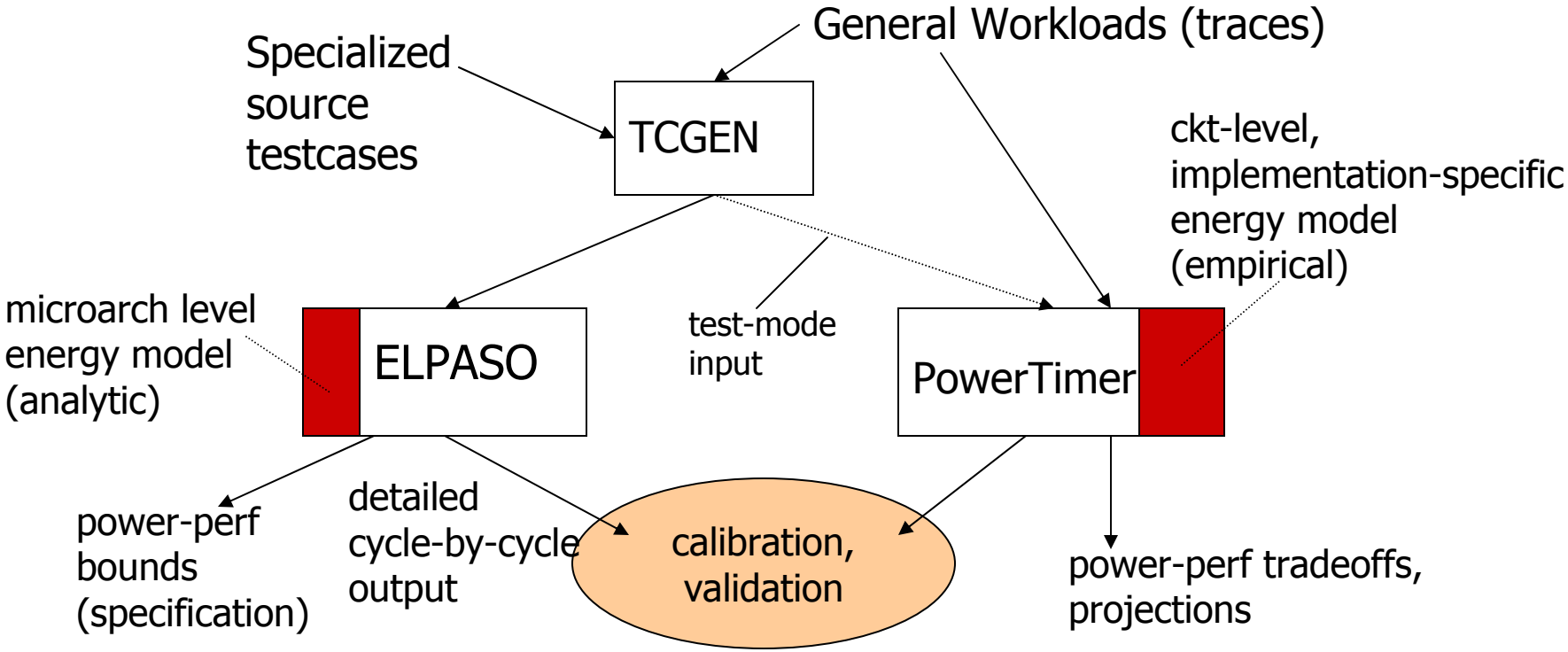
- Main challenge: **defining a specification reference**



- *Secondary problems:*
  - *generate apt test cases*
  - *test case coverage*
  - *choice of o/p signatures*

The Elpaso Reference Model

- Early Stage Power/Perf Analysis, Specification, Optimization
  - > a validation reference model
  - > tescase suite used will be part of next release of PowerTimer



More recently: a trace analysis tool called Trance is also being used for cross-calibration purposes, in addition to the elpaso cycle-accurate reference

Bounding Perf and Power

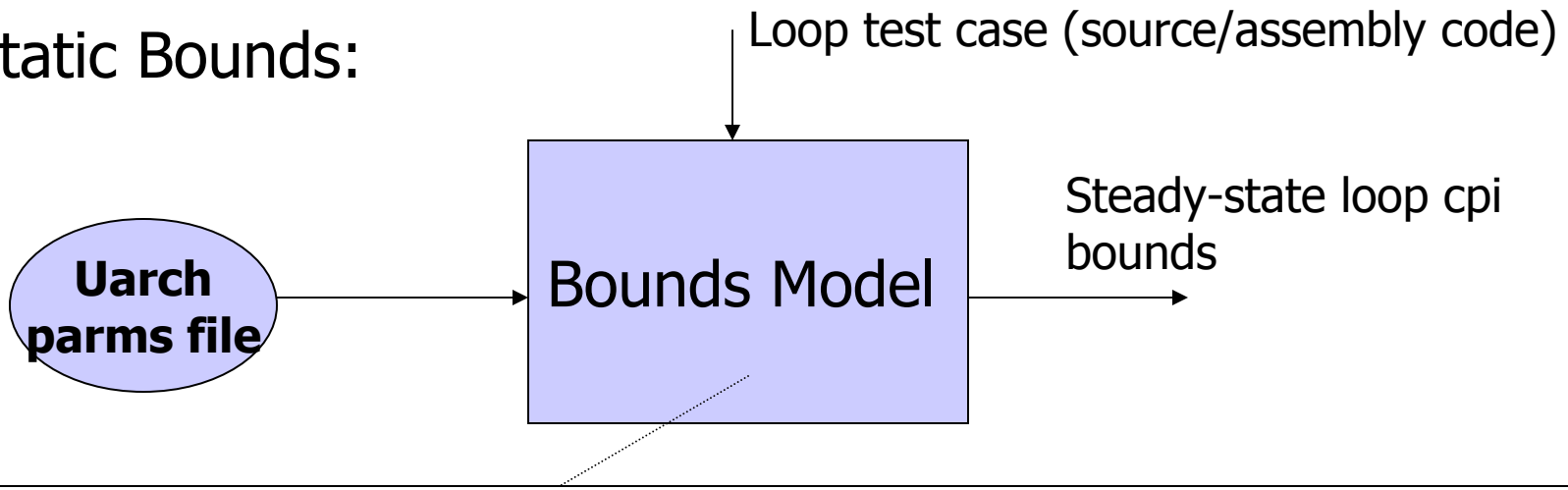
- Lower and upper bounds on expected model outputs can serve as a viable "spec", in lieu of an exact reference
- Even a single set of bounds (upper or lower) is useful
- Utilization/power bounds based on IPC bounds are also predicatable, using prior pipeline flow-based theory

Test Case Number	Performance Bounds				Utilization/Power Bounds	
	Cpi (ub)	Cpi (lb)	T (ub)	T(lb)	Upper bound	Lower bound
TC.1						
TC.2						
. .						
TC.n						

regression test suite – used in testing model versions

Performance Bounds

- Static Bounds:



*** IBM Research, Bose et al. 1995 - 2000: applied to perf validation for high-end PPC**
*** U of Michigan, Davidson et al. 1991 - 1998**

- Dynamic Bounds:
  - analyze a trace; build a graph; assign node/edge costs; process graph to compute net cost (time)

(e.g. Wellman96, Iyengar et al., HPCA-96)

Static Bounds - Example

```
fadd   fp3,   fp1,   fp0
lfdu    fp5,  8(r1)
lfdu   fp4,   8(r3)
fadd   fp4,   fp5,   fp4
fadd   fp1,   fp4,   fp3
stfdu  fp1,   8(r2)
bc     loop_top
```

Consider an in-order-issue super scalar machine:

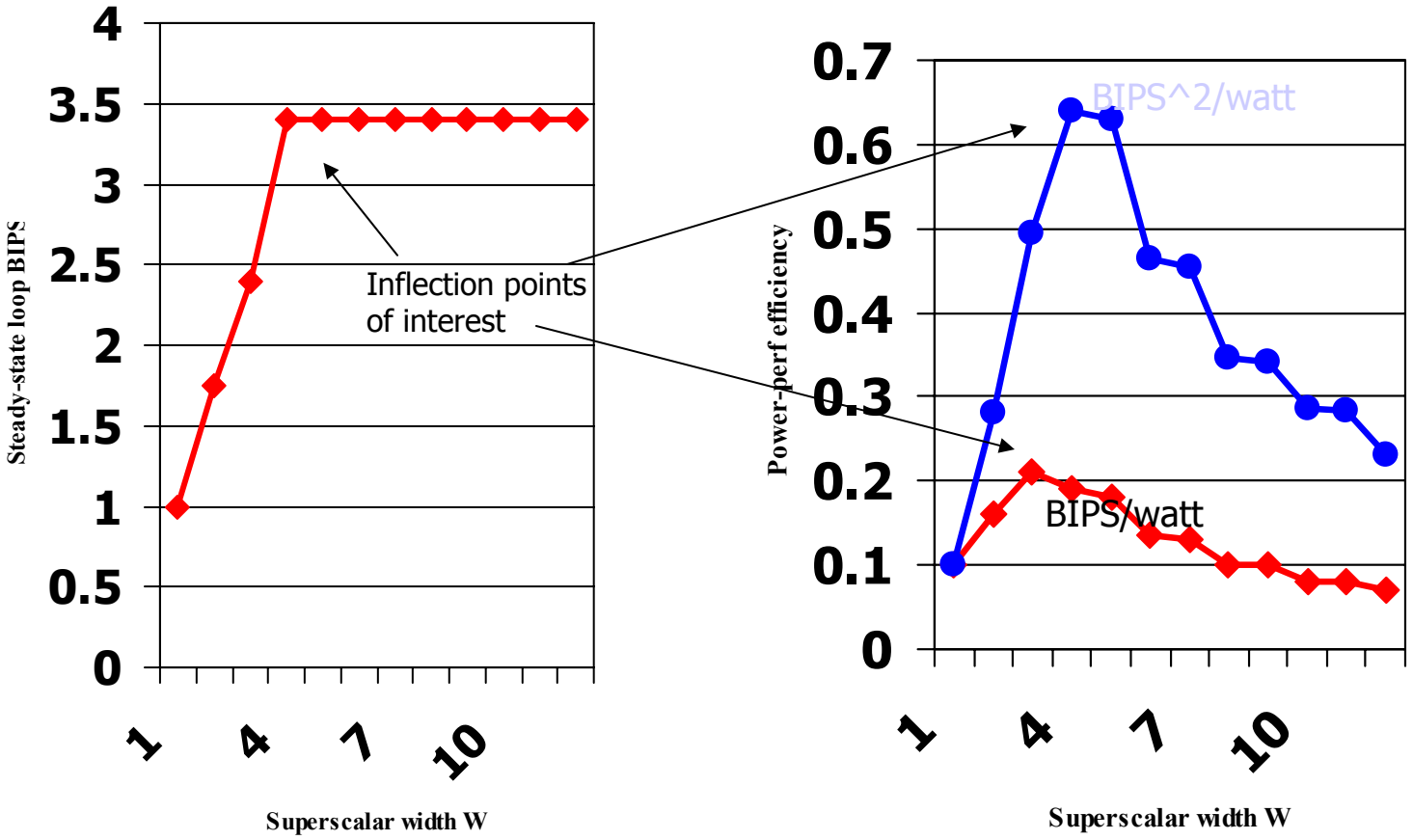
- disp_bw = iss_bw = compl_bw = 4
- fetch_bw = 8
- l_ports = ls_units = 2
- s_ports = 1
- fp_units = 2

N = number of instructions/iteration = 7

- Steady-state loop cpi performance is determined by the narrowest (smallest bandwidth) pipe
  - above example: CPIter = 2; cpi = 2/7 = 0.286

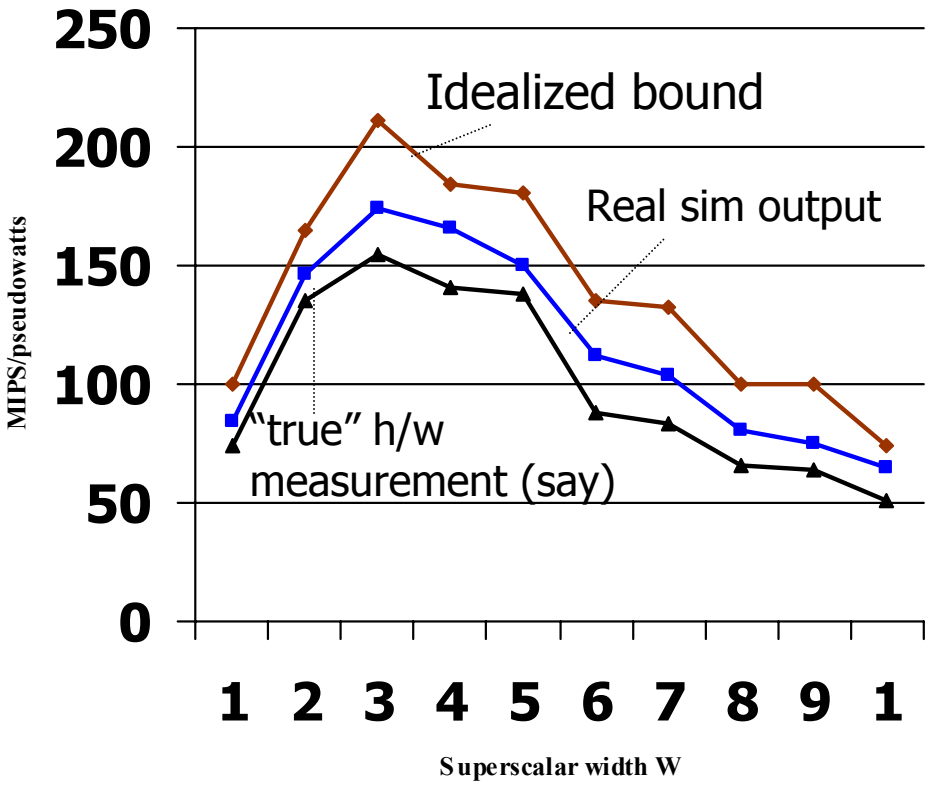
Power-Performance Bounds (El Paso)

Power ~ (W**0.5 + ls_units + fp_units + l_ports + s_ports)



(adapted from: Brooks, Bose et al. IEEE Micro, Nov/Dec 2000)

Absolute vs. Relative Accuracy



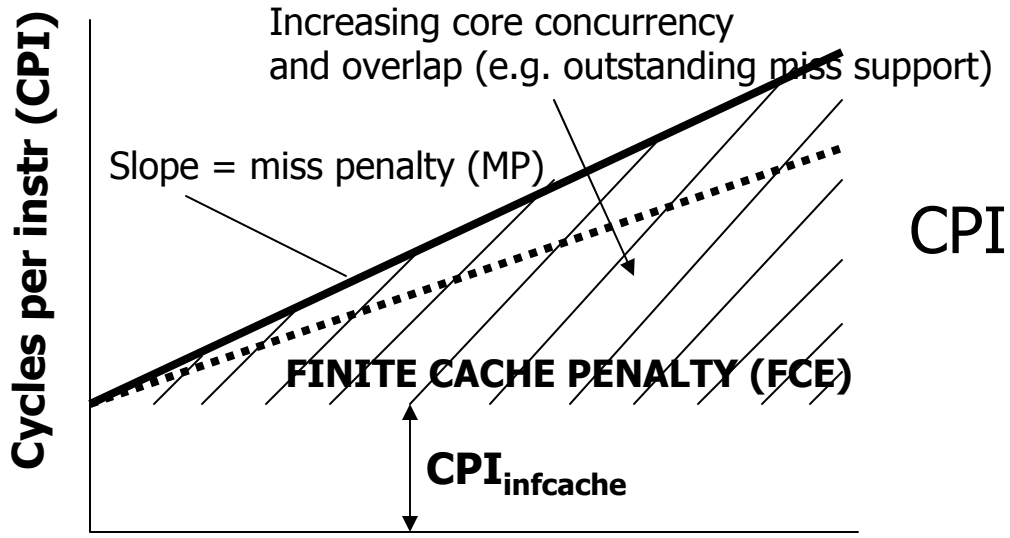
- Poor "absolute" accuracy of simulator
- But, good "relative" accuracy

In real-life, early-stage design tradeoff studies, relative accuracy is more important than absolute accuracy

Abstraction via Separable Components

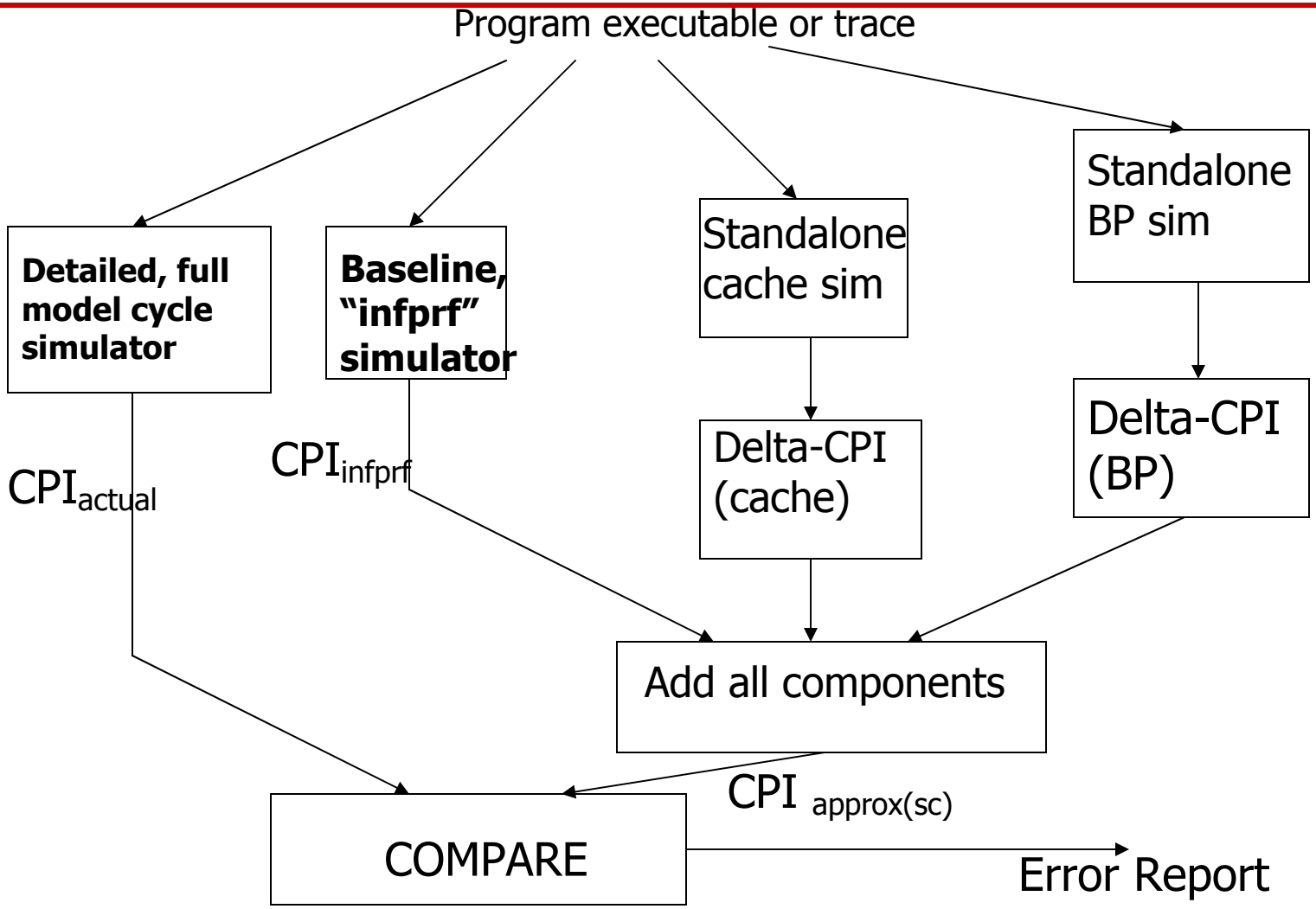
The issue of absolute vs. relative accuracy is raised in any modeling scenario: be it "performance-only", "power" or "power-performance."

Consider a commonly used performance modeling abstraction:

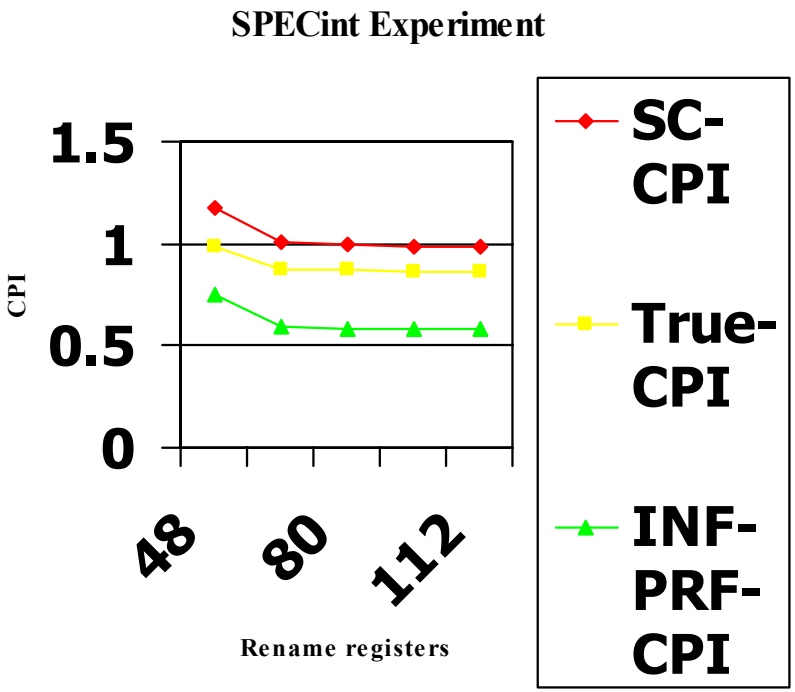


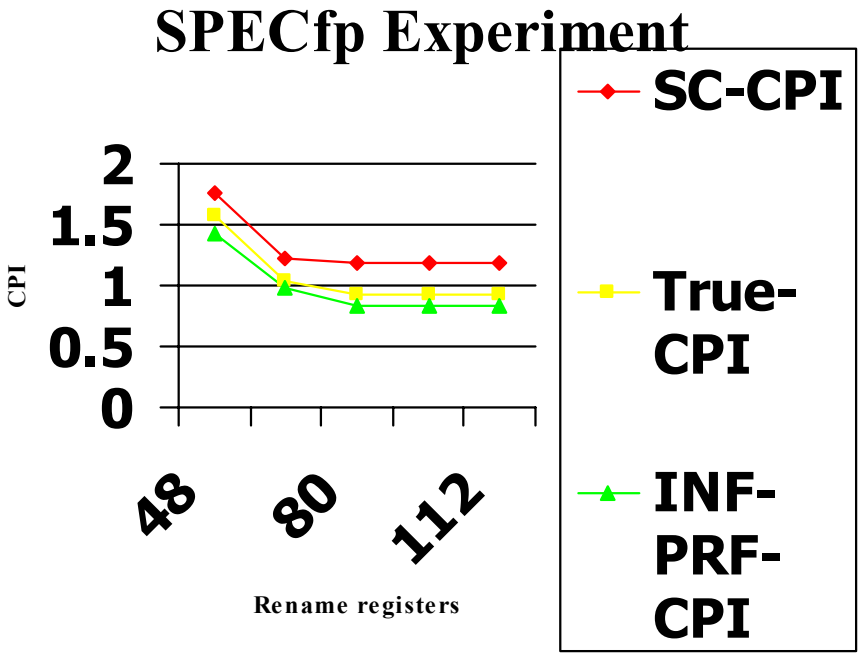
$$CPI = CPI_{infcache} + MR * MP$$

Experimental Setup: Measure Relative vs. Absolute Accuracy



Experimental Results (example)





TRUE-CPI curve: generated using PowerPC research, high-end simulator at IBM (*Turandot* simulator)

Accuracy Conclusions

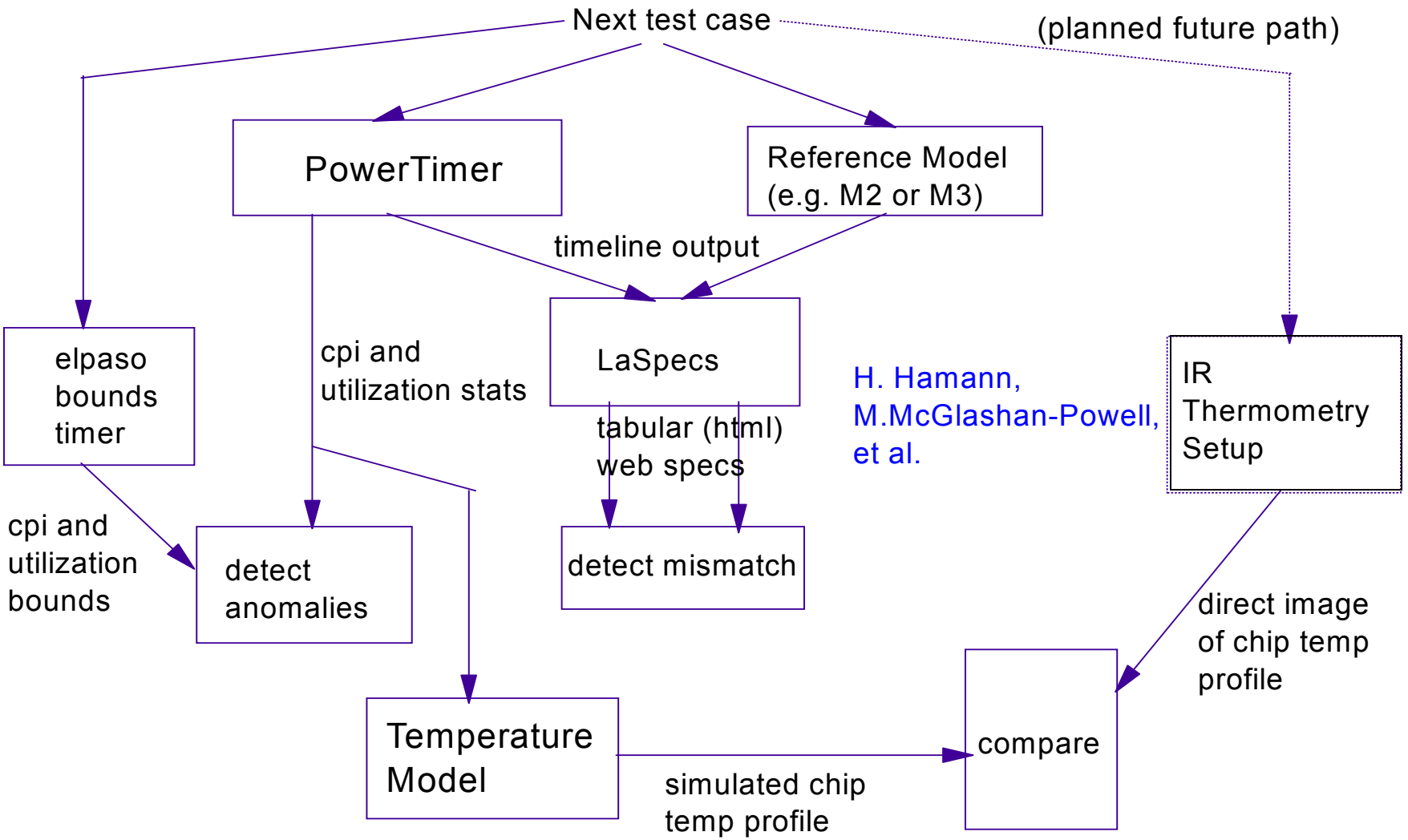
- Separable components model (for performance, and *probably* for related power-performance estimates):
  - > good for relative accuracy in most scenarios
  - > absolute accuracy depends on workload characteristics
- Detailed experiments and analysis in:

Brooks, Martonosi and Bose (2001):

"Abstraction via separable components: an empirical study of absolute and relative accuracy in processor performance modeling," IBM Research Report, Jan, 2001

Also look up Brooks' Ph.D thesis and the Wattch paper (ISCA-2000) For data on calibration of Wattch energy models against Post-layout capacitance extraction models.

Overall Validation Methodology (PowerTimer)



LaSpecs Output Example ……
see next page

Instruction Latencies

Single or Pair (Dependent) Instruction Latency (GigaProcessor)

Specifications derived from POWER4 M2 model, **version 1.222**: PowerPC opcode pair involving **lbz_add**

```
[.....F.DE.di2.h.f..c.....................................................]
[.....F.DE.d...i1.f.c.....................................................]
The instruction test case is: lbz R1,4096(R0) add R4,R1,R5
```

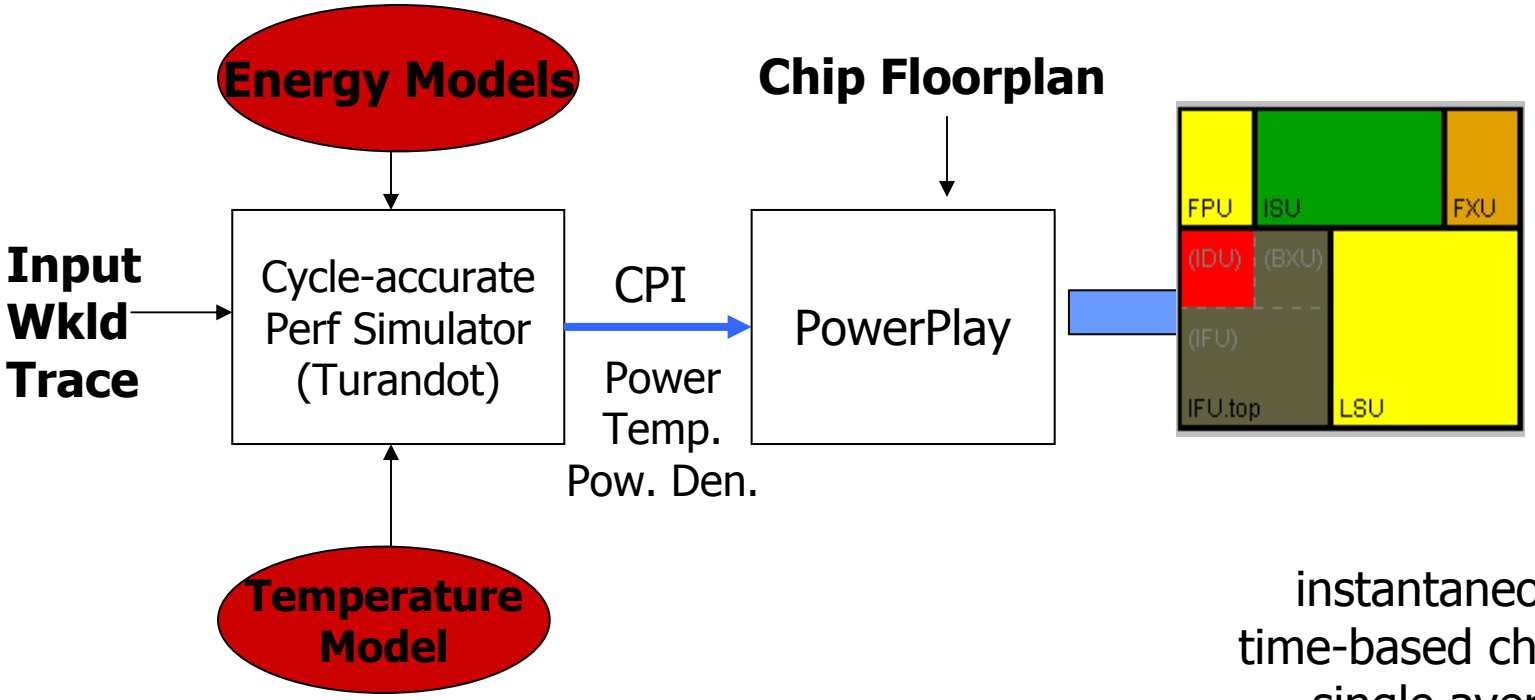
Completion Latency= 15 Live Latency= 15

Pipeline Stage	Cycle Number														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FETCH	lbz add	-	-	-	-	-	-	-	-	-	-	-	-	-	-
XMITa	-	lbz add	-	-	-	-	-	-	-	-	-	-	-	-	-
DECODE	-	-	lbz add	-	-	-	-	-	-	-	-	-	-	-	-
ASSEMBLE	-	-	-	lbz add	-	-	-	-	-	-	-	-	-	-	-
XMITb	-	-	-	-	lbz add	-	-	-	-	-	-	-	-	-	-
DISPATCH	-	-	-	-	-	lbz add	-	-	-	-	-	-	-	-	-
ISSUE	-	-	-	-	-	-	lbz	-	-	add	-	-	-	-	-
lsRREAD	-	-	-	-	-	-	-	lbz	-	-	-	-	-	-	-
EXECUTE	-	-	-	-	-	-	-	-	lbz	-	-	add	-	-	-
FINISH	-	-	-	-	-	-	-	-	-	-	-	lbz	add	-	-
XMITc	-	-	-	-	-	-	-	-	-	-	-	-	lbz	add	-
COMPLETE	-	-	-	-	-	-	-	-	-	-	-	-	-	-	lbz add

All single and pair instruction latency specs were matched Turandot vs. Reference M2 model

Other Visualization Aids for Validation and Calibration….

PowerPlay: a Floorplan Visualizer



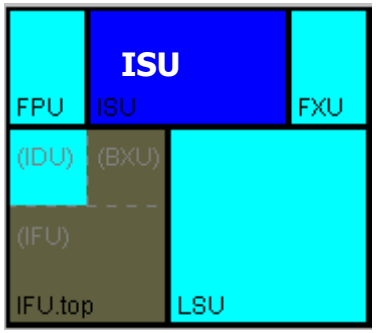
instantaneous,
time-based changes,
single average

- Java-based
- Currently operational: works with MET/Turandot (PowerTimer) toolset

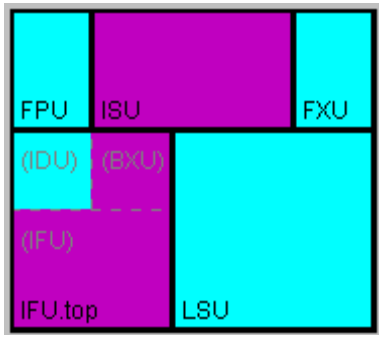
Implementation: by Rose Liu, summer intern

PowerPlay Example/Demo

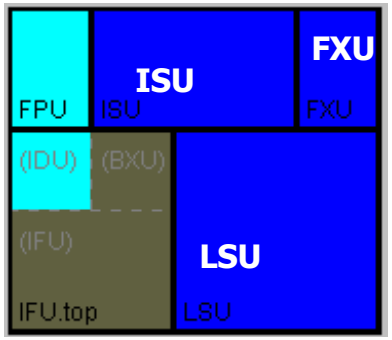
Power Density Characteristics (SPEC2K)



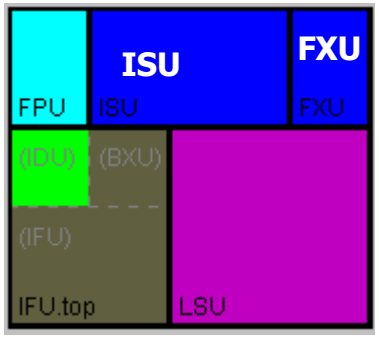
Gap – 3.13



Perlbmk – 3.14



Twolf – 2.69



Art – 2.45

- Simple (illustrative) wkld characterization metric:

$$K = (1/Area_{total}) \Sigma C_i Area_i \text{ , (cool)} 1 \leq C_i \leq 9 \text{ (hot)}$$

Calibration of Temperature Models

- Current aids:
  - Infra-red based thermometry set up
  - On-chip temperature sensors
- Test cases run on simulator and on hardware
  - Simulated results vs. direct measurements
  - Current measurements available for recent product chips: could not be shown due to clearance difficulty

Tutorial Outline

	Introduction and Motivation
	Basics of Performance Modeling
	- Turandot performance simulation infrastructure
	Architectural Power Modeling
	- PowerTimer extensions to Turandot
	- Power-Performance Efficiency Metrics (Victor)
	Case Studies and Examples
	- Optimal Power-Performance Pipeline Depth
	Validation and Calibration Efforts
11:45-12:00	Future challenges and Discussion
	Bibliography