Teradactyl:
An Easy-to-Use Supercomputer

Augustus K. Uht
Dept. of Electrical and Computer Engineering

UNIVERSITY OF
Rhode Island

@URI
μURI
Microarchitecture Research Institute
University of Rhode Island

SSCCII 2004: January 30

Derivation

- From: "*Pterodactyl*" – winged fingers
  - Flying dinosaur
- ***Teradactyl***: 'Flying' Supercomputer
  - Many 'fingers'
  - Each 'finger' (processing element) flies

Acknowledgement

Work supported in present, past and future by:

- Laurette Bradley, my wife.

Outline

1. The Problem
2. Our Approach to a Solution
3. Resource-Flow Execution
4. Basic LEVO Microarchitecture
5. Teradactyl Architecture
6. Summary

THE Supercomputing Problem

- PROGRAMMABILITY!!!
- Why? Here's why:

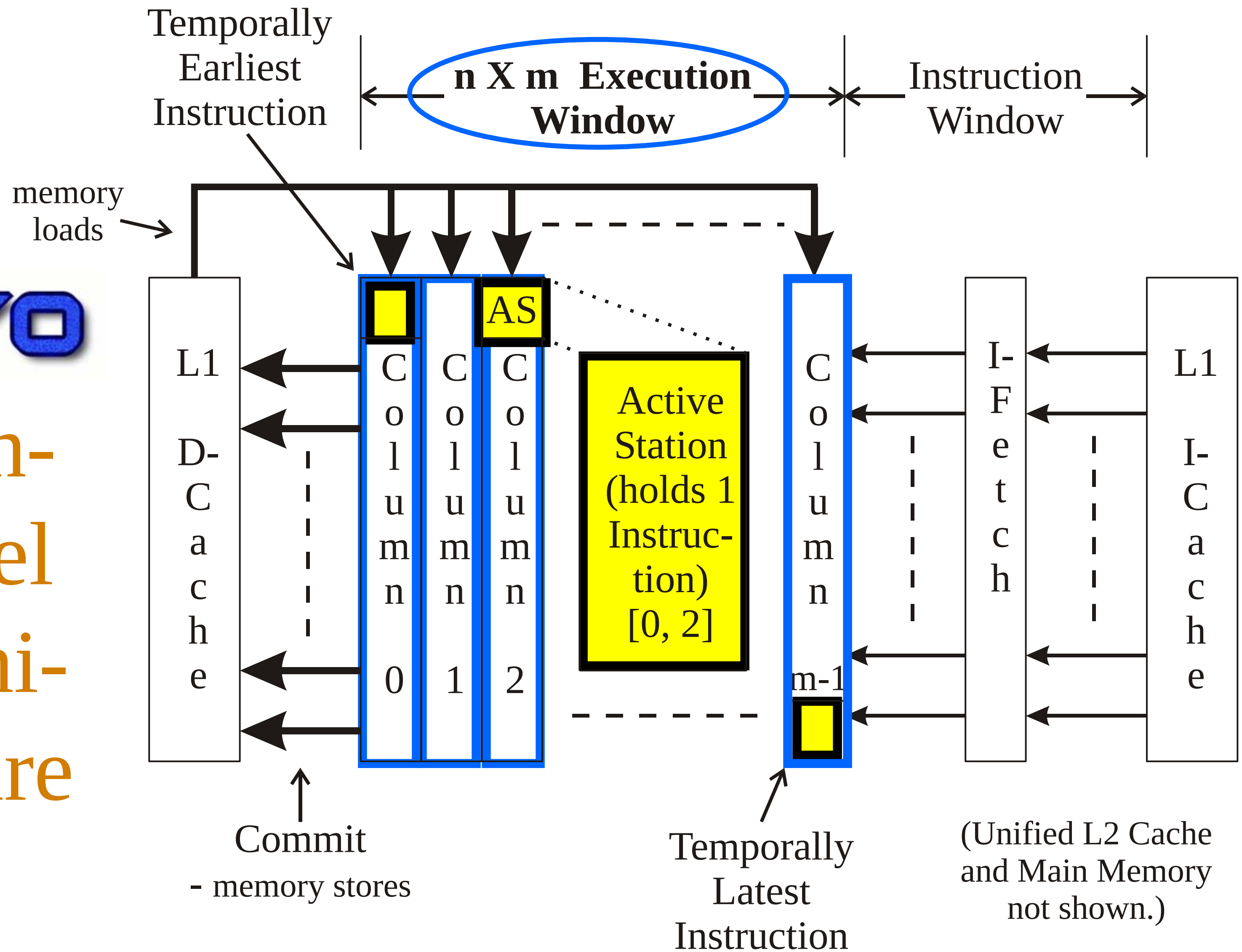
1. Scientific programs by users take years to write
   - Even with libraries

     "…the manual development and testing of a reasonably efficient parallel code for a computational model … typically takes ***months to years*** for a computational chemist." (Our emphases.), Supercomputing 2002
   - Parallel programming, scheduling, etc. way too hard
2. Users' time is greatly misspent:
   chemists should be doing Chemistry, not coding

Our Approach to a Solution

1. Use ***hardware*** to do the hard stuff
2. Let the user use an ***easy*** programming model:
   → ***Standard sequential (imperative) model***
3. WHY is parallel programming so hard?
   → don't know where the ***data dependencies*** are
4. Approach has always been to estimate them
5. Don't <u>estimate</u> them, <u>know</u> them
6. Use ***resource-flow*** execution:
   - Instructions flow to PE's,
     are executed <u>regardless</u> of dependencies
   - Then clean up: <u>enforce</u> dependencies when they're <u>known</u>

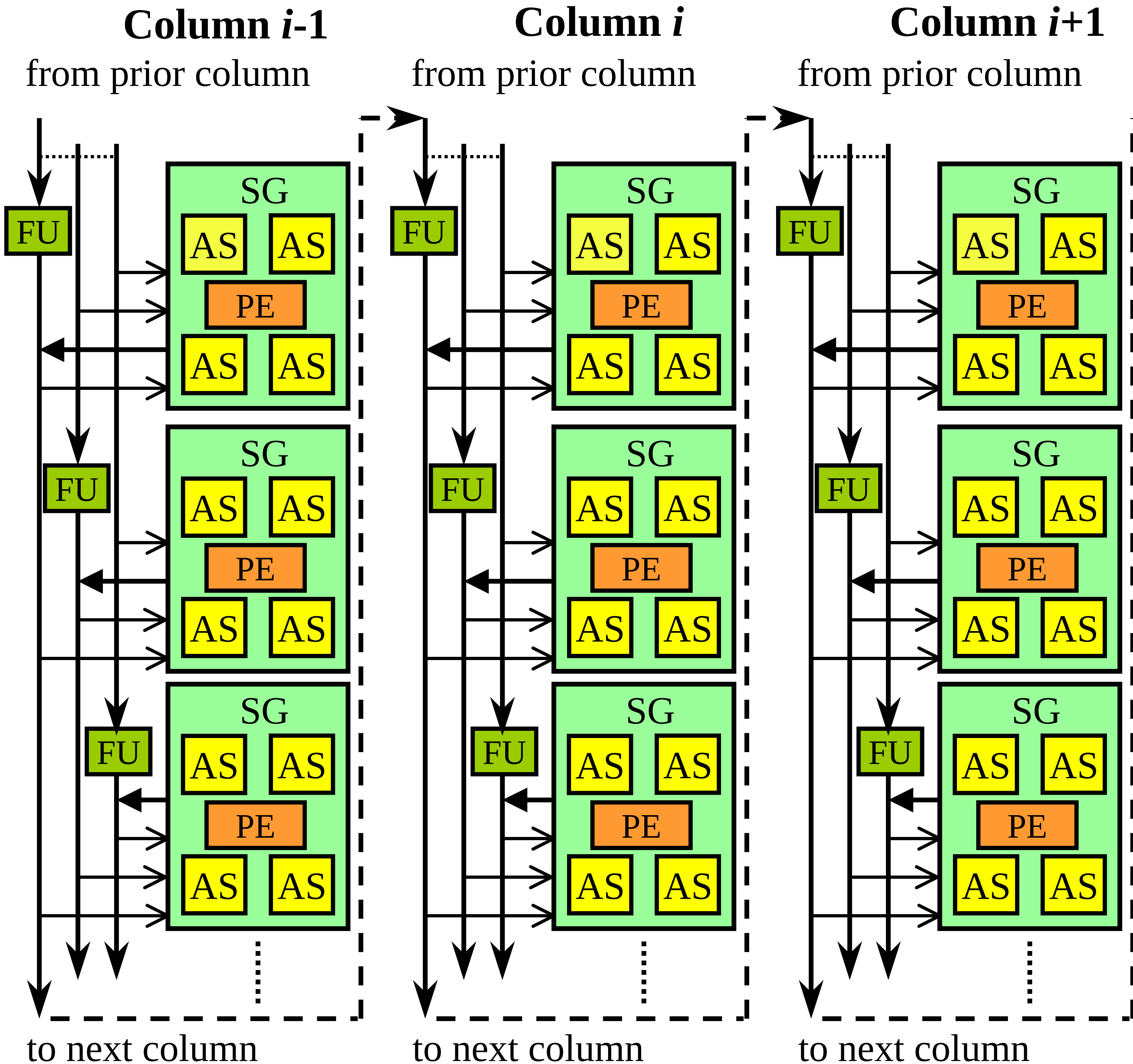
High-Level Architecture



Processing Elements (PEs) are distributed among AS's.

Micro-architecture

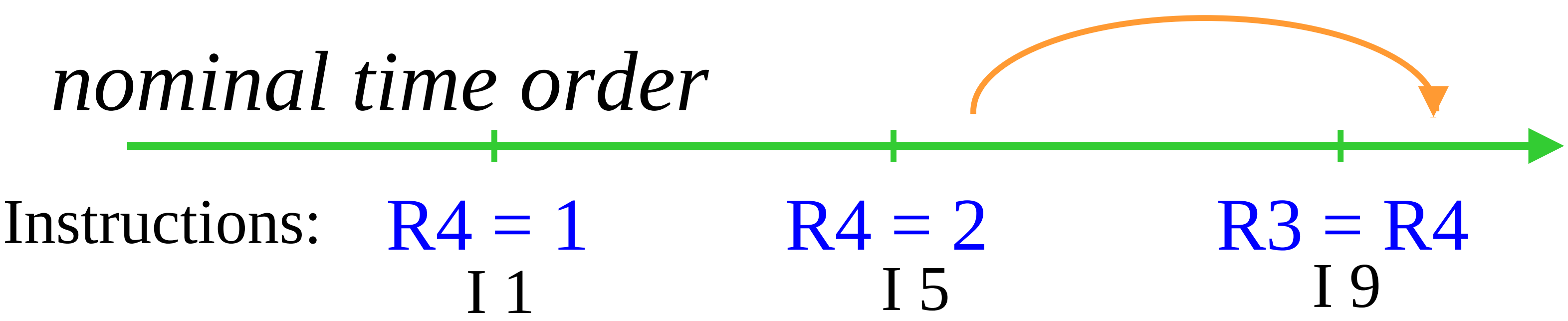
(Execution Window)



SG: Sharing Group

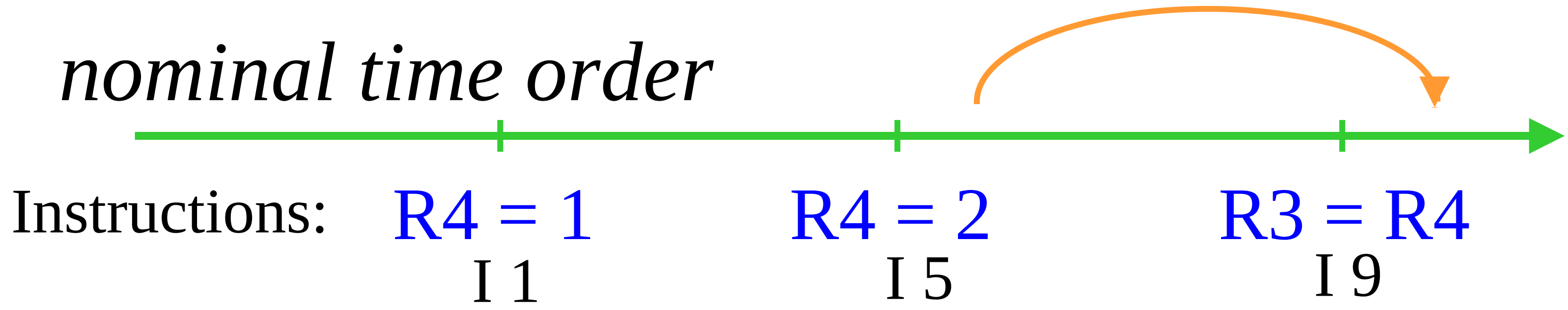
Time-Tags

- Small integers = position in E-window
- Enforce and minimize dependencies
- Provide operand linking (sink-to-source)
- Basic problem:
  - R3 must wind up with *closest previous value* of R4 (2)
  - Must be independent of execution order of instructions



Time-Tag Example: Case 1

- LSTT: Last Snarfed Time-Tag



<u>Time – 1</u>: I 1 brdcsts.

R4 address matches,
TT(I 1) >= LSTT(I 9),
I 1 info snarfed: R3=1

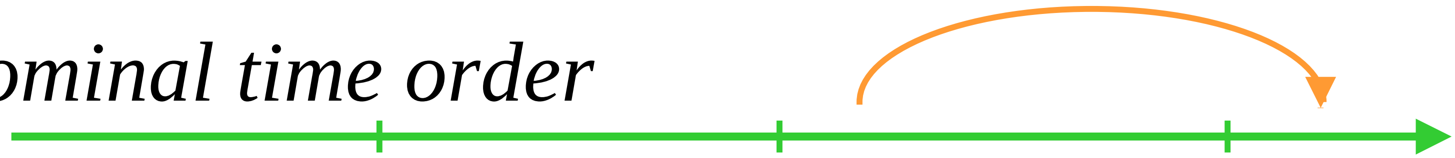
<u>Time – 2</u>: I 5 brdcsts.

R4 address matches,
TT(I 5) >= LSTT(I 9),
I 5 info snarfed: R3=**<u>2</u>**

Time-Tag Example: Case 2

- Recall: R3 ← *closest previous value* of R4 (2)

nominal time order



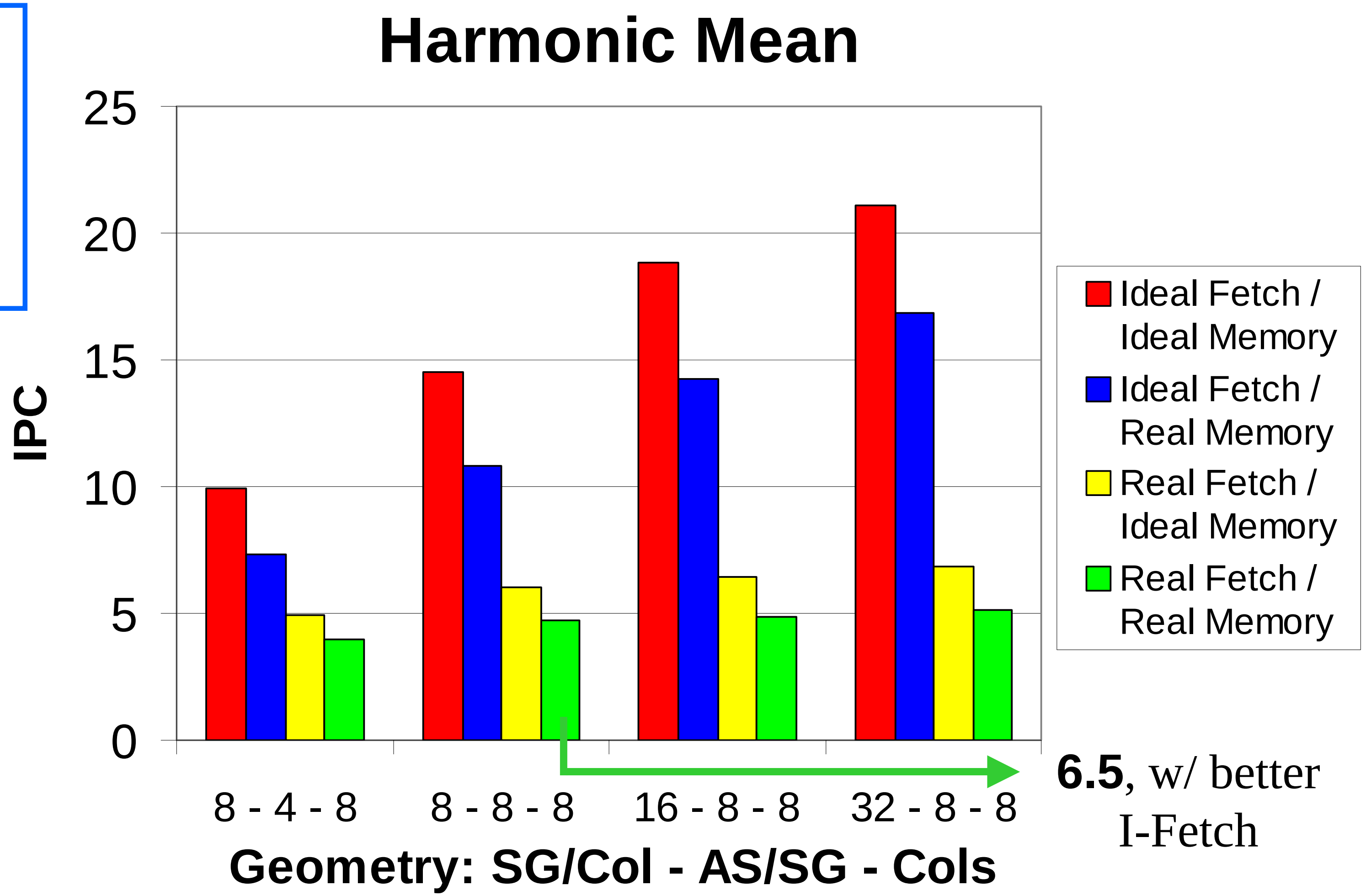
Instructions: R4 = 1 (I 1) R4 = 2 (I 5) R3 = R4 (I 9)

Time – 1: I 5 brdcsts. R4 address matches, TT(I 5) >= LSTT(I 9), I 5 info snarfed: R3=**2**

Time – 2: I 1 brdcsts. R4 address matches, TT(I 1) < LSTT(I 9), I 1 info **not** snarfed.

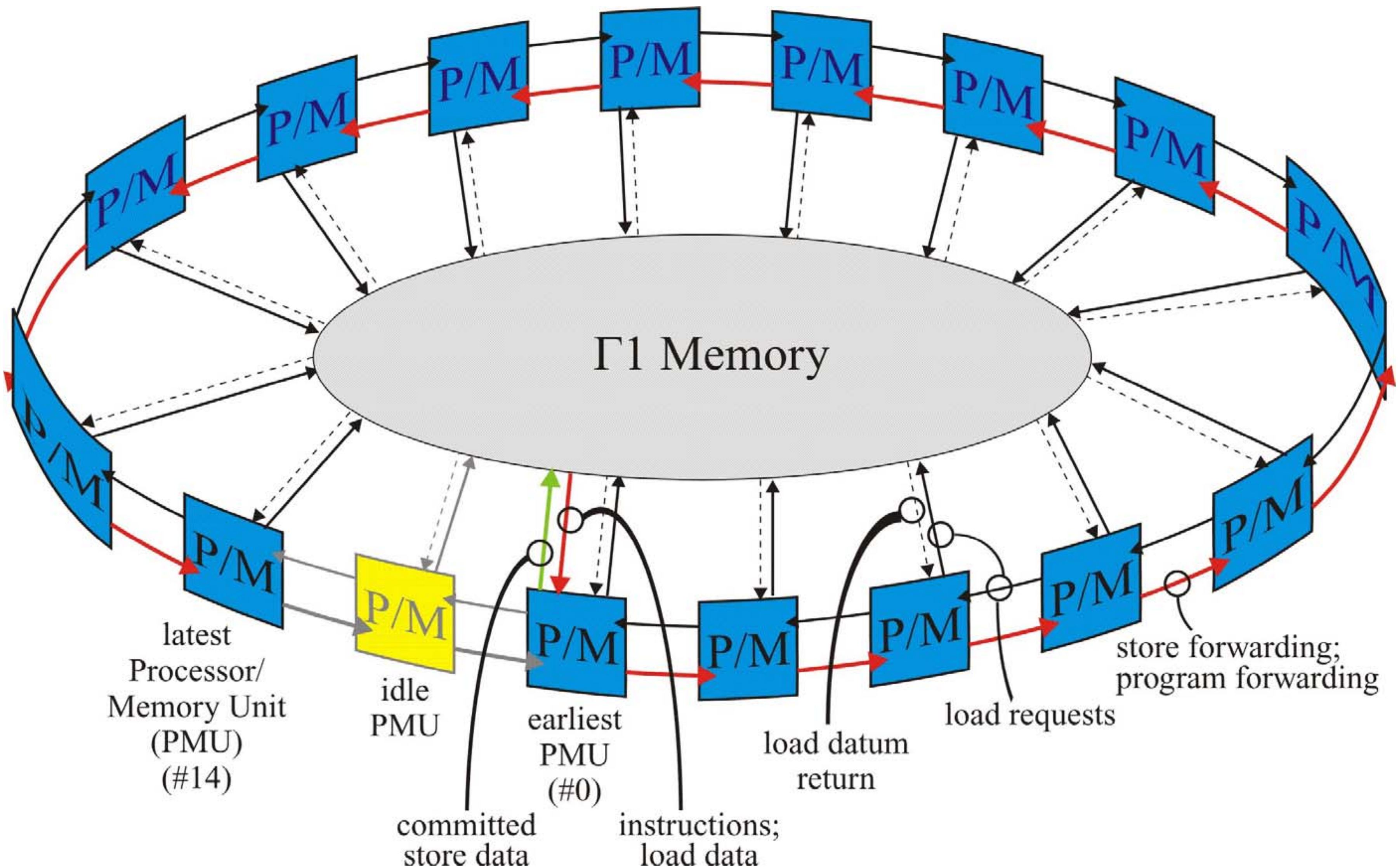
LEVO Performance

On hard code: SPECInt



On <u>easy</u> code: ??

Teradactyl



Γ1 Memory
P/M
latest Processor/ Memory Unit (PMU) (#14)
idle PMU
earliest PMU (#0)
committed store data
instructions; load data
load datum return
load requests
store forwarding; program forwarding

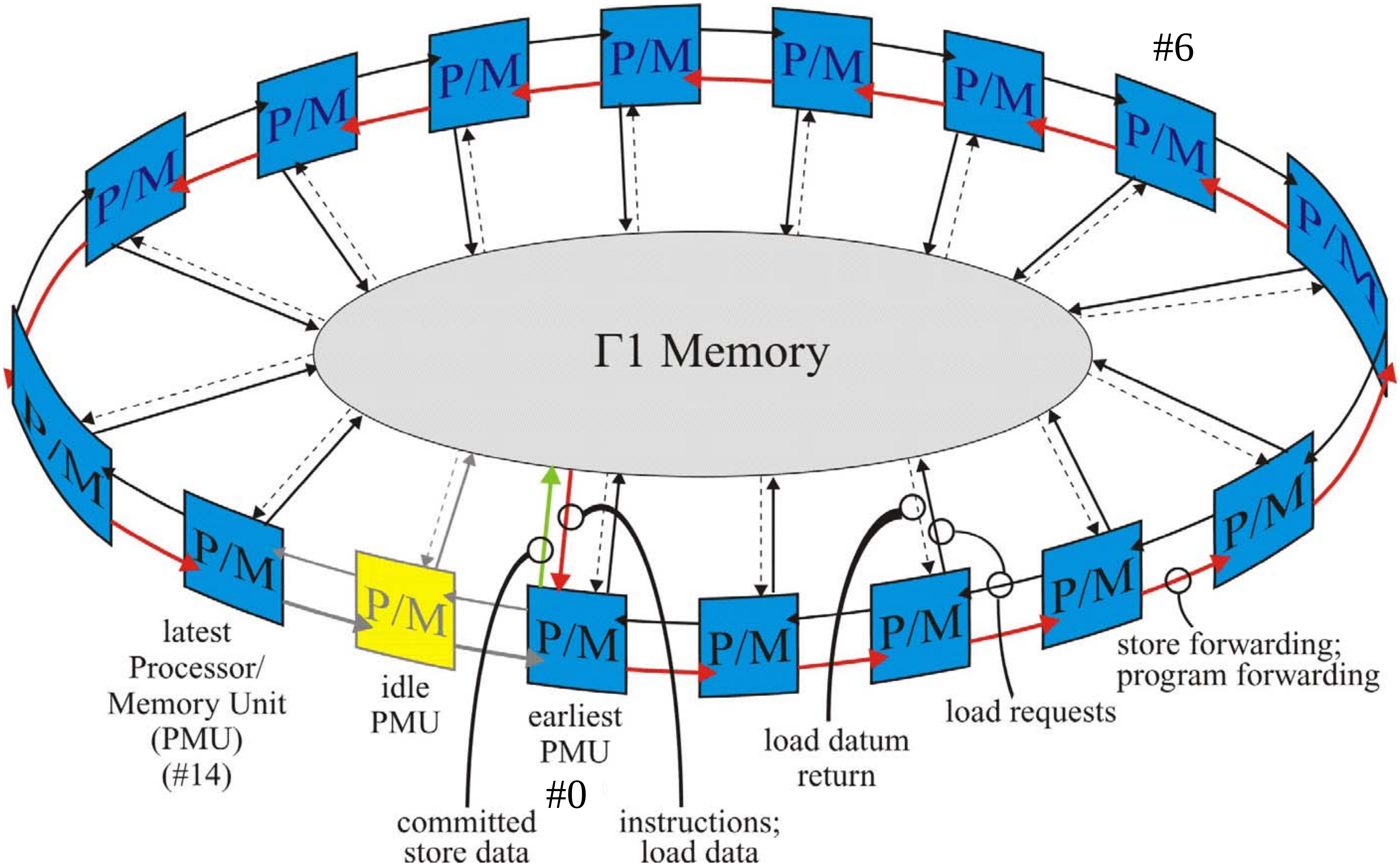
Teradactyl

Example

(A x B) + C

Before: ‘+’ (#6) wait for ‘x’ (#0); manual

After: ‘+’ (#6) || ‘x’ (#0); & auto-schedule



Teradactyl Characteristics

- Scalable to thousands of processors
- Uses modified LEVO model:
  - *Processor Memory Units* like Levo columns
    - Augment time-tags with PMU # as a prefix
  - Whole Teradactyl like Levo Execution Window
- Once data computed, is sent around ring, to update dependent operands:
  → close to best performance possible
  (→ with speculation, maybe better)

Teradactyl Performance

- First: Talking about SUSTAINED performance
- Now, assume:
  - PMU (Levo) up to 10 IPC within several years
  - Chip clock frequency up to 5 GHz → 50 Gops
  - Then for TeraOp: ~25 PMU's (some inefficiency)
  - And for PetaOp: 25,000 PMU's
- Power, etc.: ~ same as other supercomputers
- Other supercomputers: ~< 1 TeraOp, sustained
  - With 100's or 1000's of processors

Summary

- Problem: Programmability
- Solution: Teradactyl:
  - Based on resource-flow execution
  - Data dependencies known exactly, at run-time
  - Data speculation also used to improve performance
  - Scalable
  - "Easy" to program (well, as easy as it can be :-)
- The future: Petadactyl

Teradactyl:
An Easy-to-Use Supercomputer

Augustus K. Uht
Dept. of Electrical and Computer Engineering

SSCCII 2004: January 30