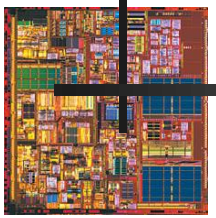
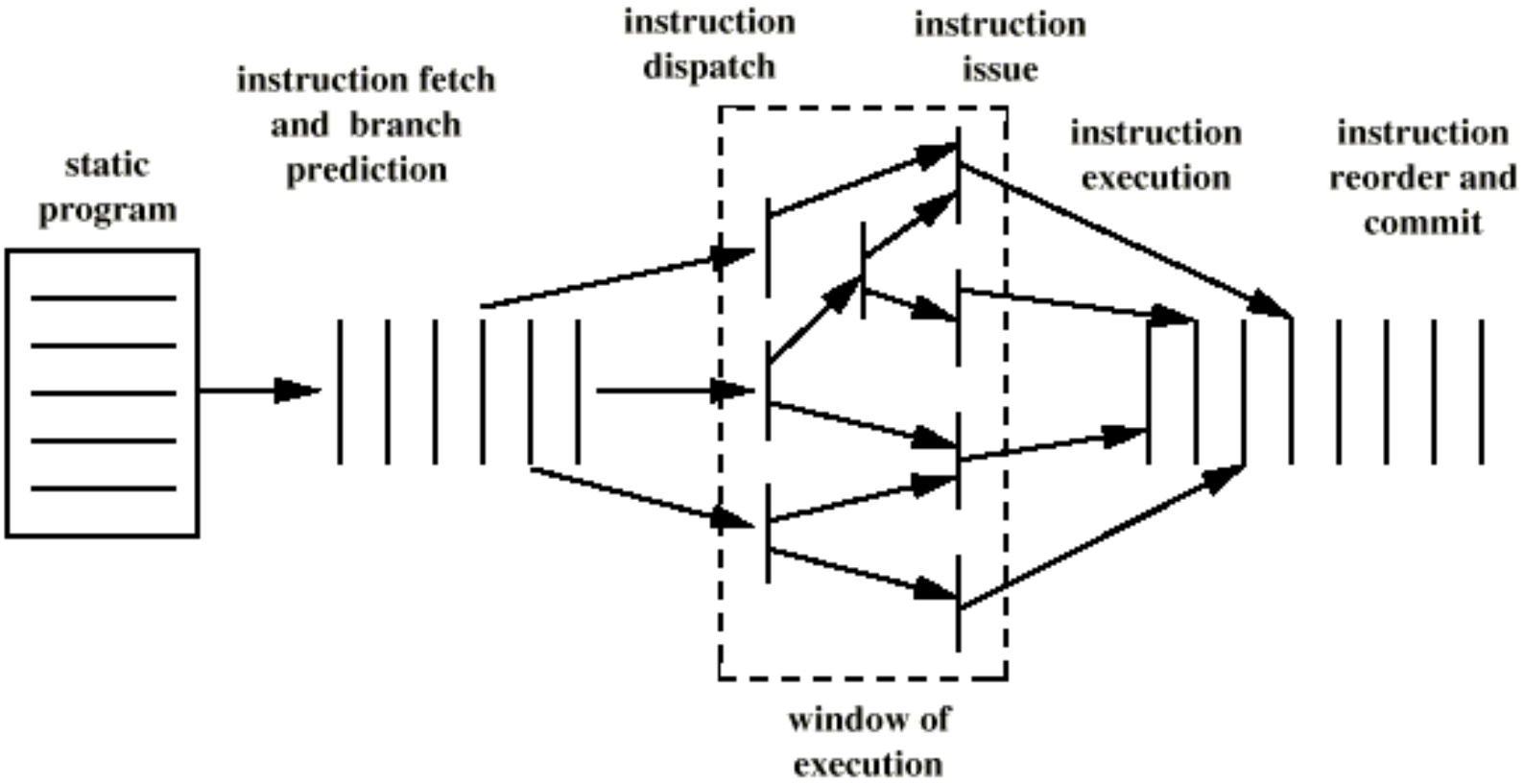


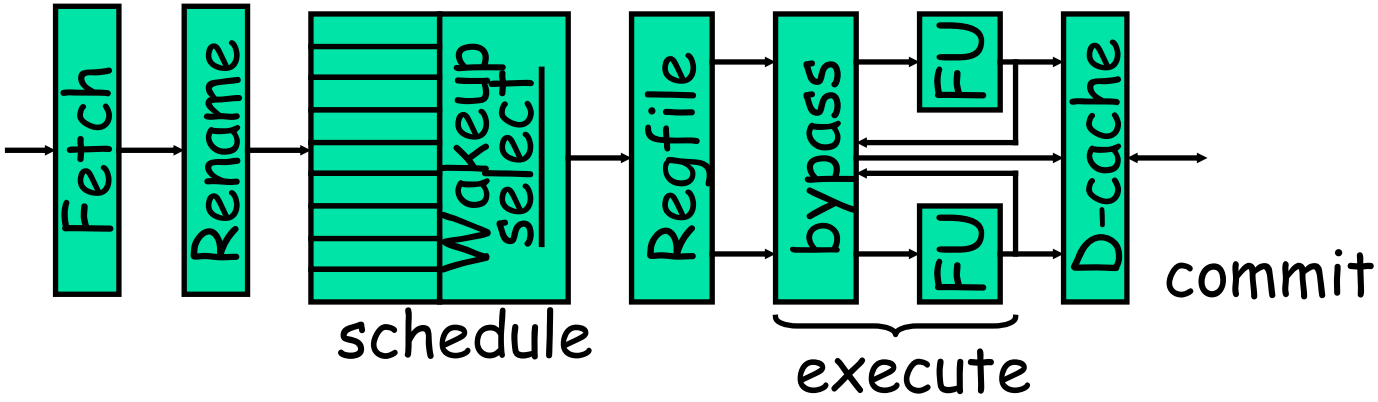
Superscalar Complexity

View of Superscalar Execution

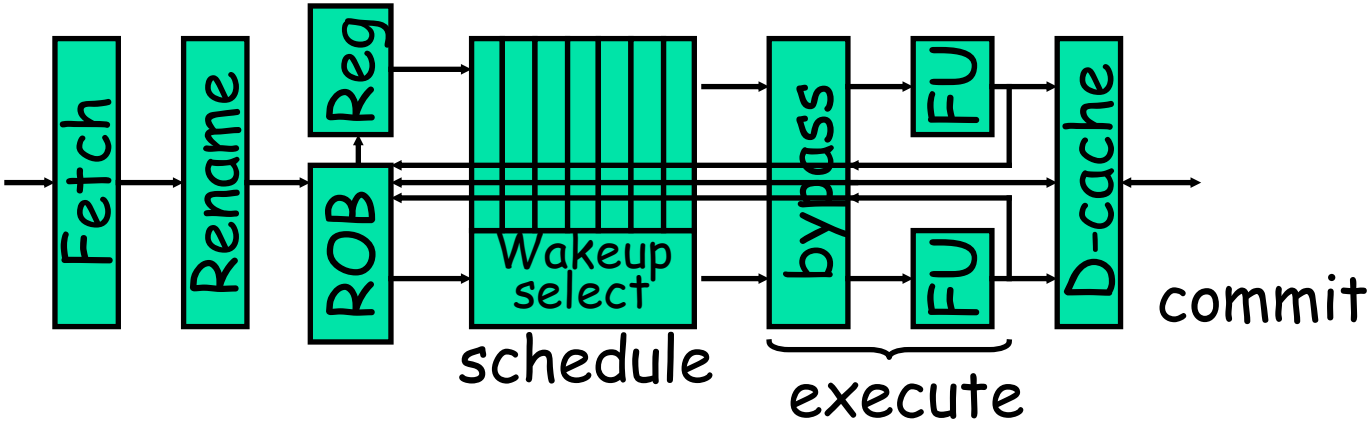


Generic Superscalar Processor Models

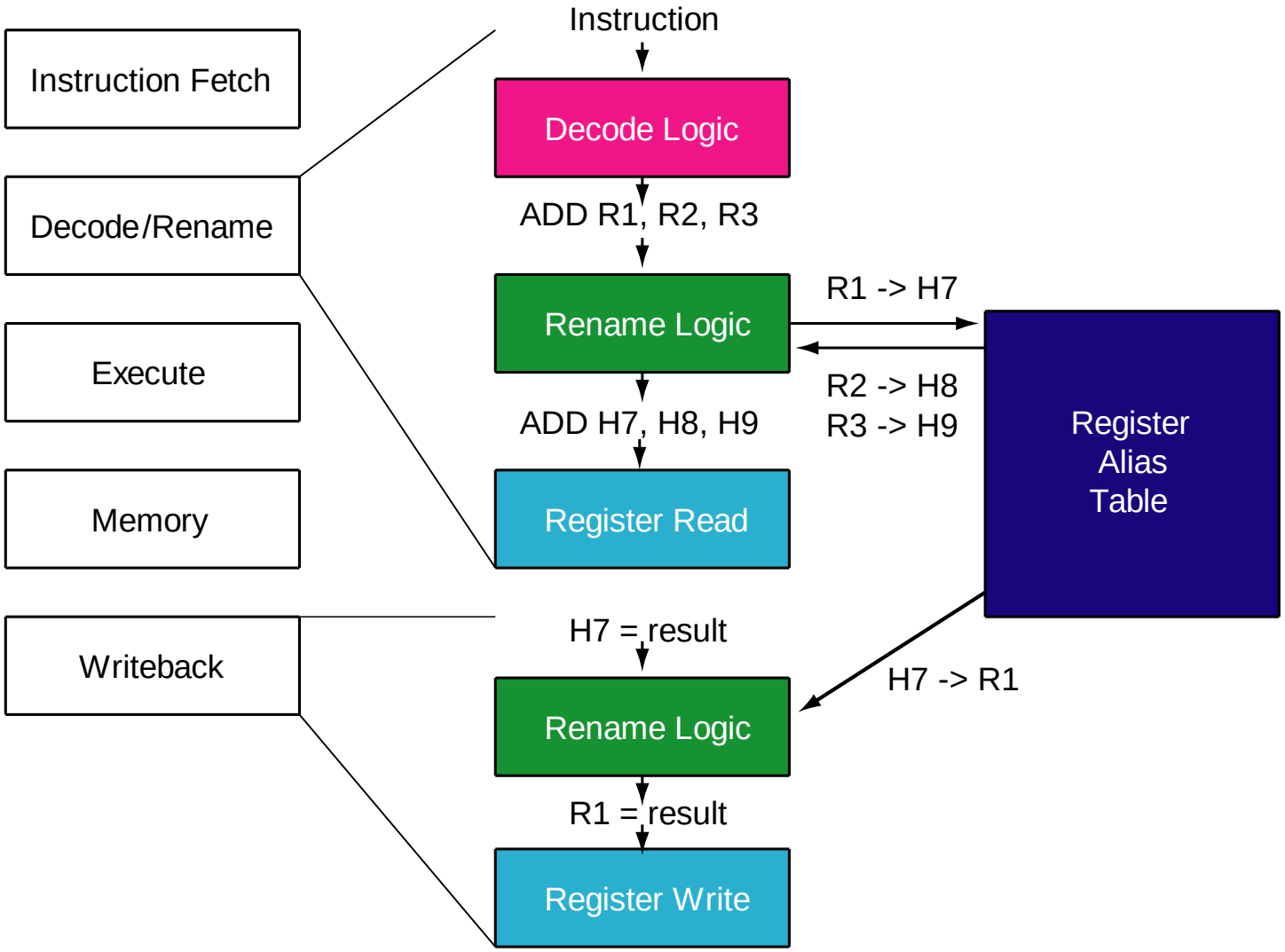
Issue queue based

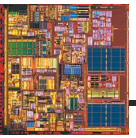


Reservation based



Register Renaming





Register Renaming

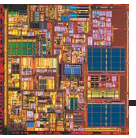
- Alpha 21264+, MIPS R10K+, Pentium 4 use explicit register renaming
  - Registers are not read until instruction dispatches (begins execution)
  - Register renaming ensures no conflicts

```
DIV R5, R4, R2
ADD R7, R5, R1
SUB R5, R3, R2
LW  R7, 1000(R5)
```

RAT

R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR13
R6	PR20
R7	PR30

...



Register Renaming

- Assume that PR37 is free and allocated to DIV's destination register R5

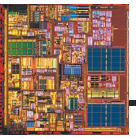
```
DIV R5, R4, R2
ADD R7, R5, R1
SUB R5, R3, R2
LW  R7, 1000(R5)
```

```
DIV PR37, PR45, PR2
```

RAT

R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	~~PR13~~ PR37
R6	PR20
R7	PR30

...



Register Renaming

- Assume that PR37 is free and allocated to DIV's destination register R5

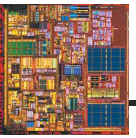
```
DIV R5, R4, R2
ADD R7, R5, R1
SUB R5, R3, R2
LW  R7, 1000(R5)
```

```
DIV PR37, PR45, PR2
```

RAT

R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR37
R6	PR20
R7	PR30

...



Register Renaming

- Assume that PR4 is free and allocated to ADD's destination register R7

```
DIV R5, R4, R2
ADD R7, R5, R1
SUB R5, R3, R2
LW  R7, 1000(R5)
```

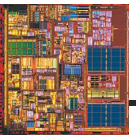
```
DIV PR37, PR45, PR2
ADD PR4 , PR37, PR23
```

RAT

R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR37
R6	PR20
R7	~~PR30~~

PR4

...



Register Renaming

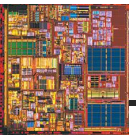
```
DIV R5, R4, R2
ADD R7, R5, R1
SUB R5, R3, R2
LW  R7, 1000(R5)
```

```
DIV PR37, PR45, PR2
ADD PR4 , PR37, PR23
SUB PR42, PR17, PR2
```

RAT

R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	~~PR37~~ PR42
R6	PR20
R7	PR4

...



Register Renaming

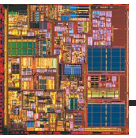
```
DIV R5, R4, R2          DIV PR37, PR45, PR2
ADD R7, R5, R1          ADD PR4 , PR37, PR23
SUB R5, R3, R2          SUB PR42, PR17, PR2
LW  R7, 1000(R5)        LW  PR19, 1000(PR42)
```

RAT

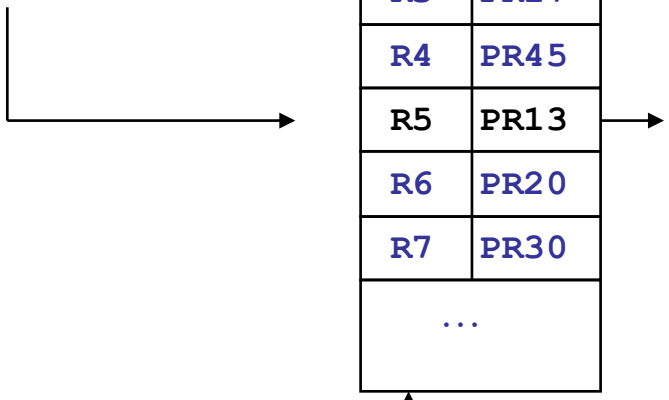
R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR42
R6	PR20
R7	~~PR4~~ PR19

...

MIPS R10000, Recycling Physical Registers

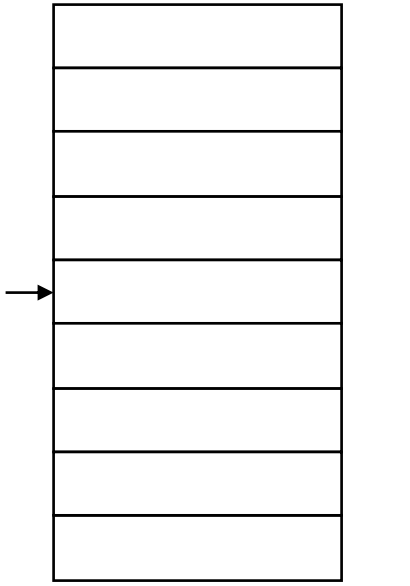


```
I1:DIV R5, R4, R2
I2:ADD R7, R5, R1
I3:SUB R5, R3, R2
I4:LW  R7, 1000(R5)
```

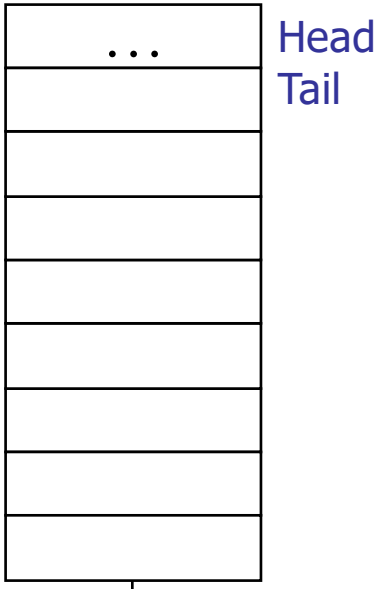


Register Map

R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR13
R6	PR20
R7	PR30
...	



Instruction Queue
(Reservation Stations)



Active List

...

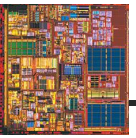
Head
Tail

Register Free List

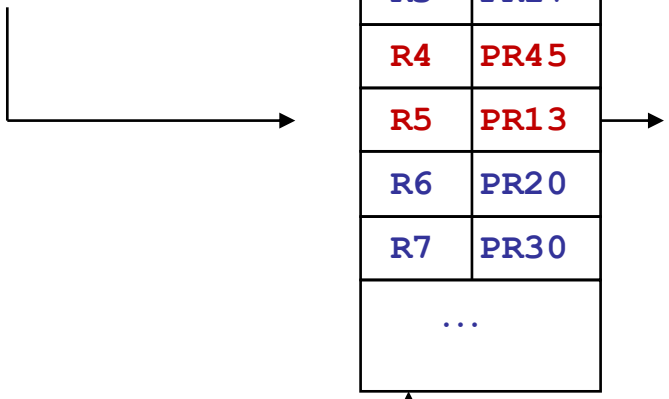
PR37, PR4, PR42, PR19, ...

Commit

MIPS R10000, Recycling Physical Registers

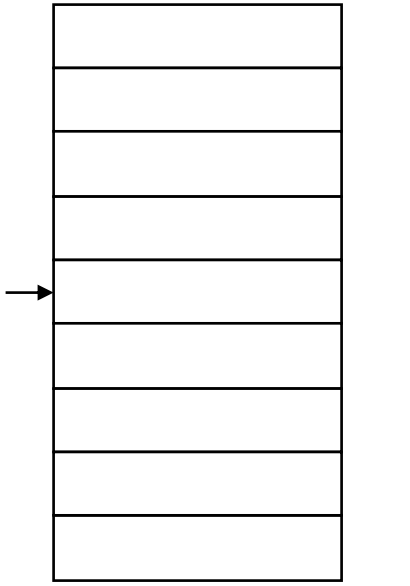


```
I1:DIV R5, R4, R2
I2:ADD R7, R5, R1
I3:SUB R5, R3, R2
I4:LW  R7, 1000(R5)
```

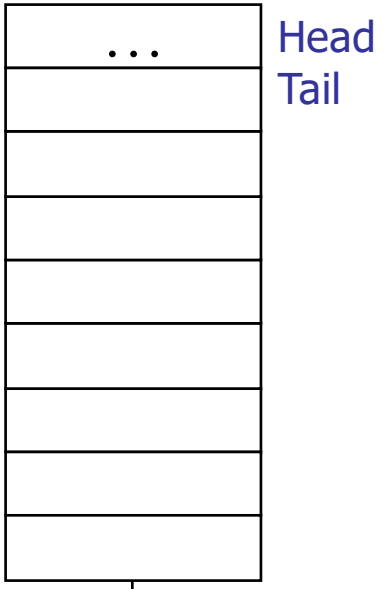


Register Map

R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR13
R6	PR20
R7	PR30
...	



Instruction Queue
(Reservation Stations)



Active List

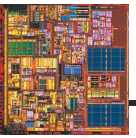
...

Head
Tail

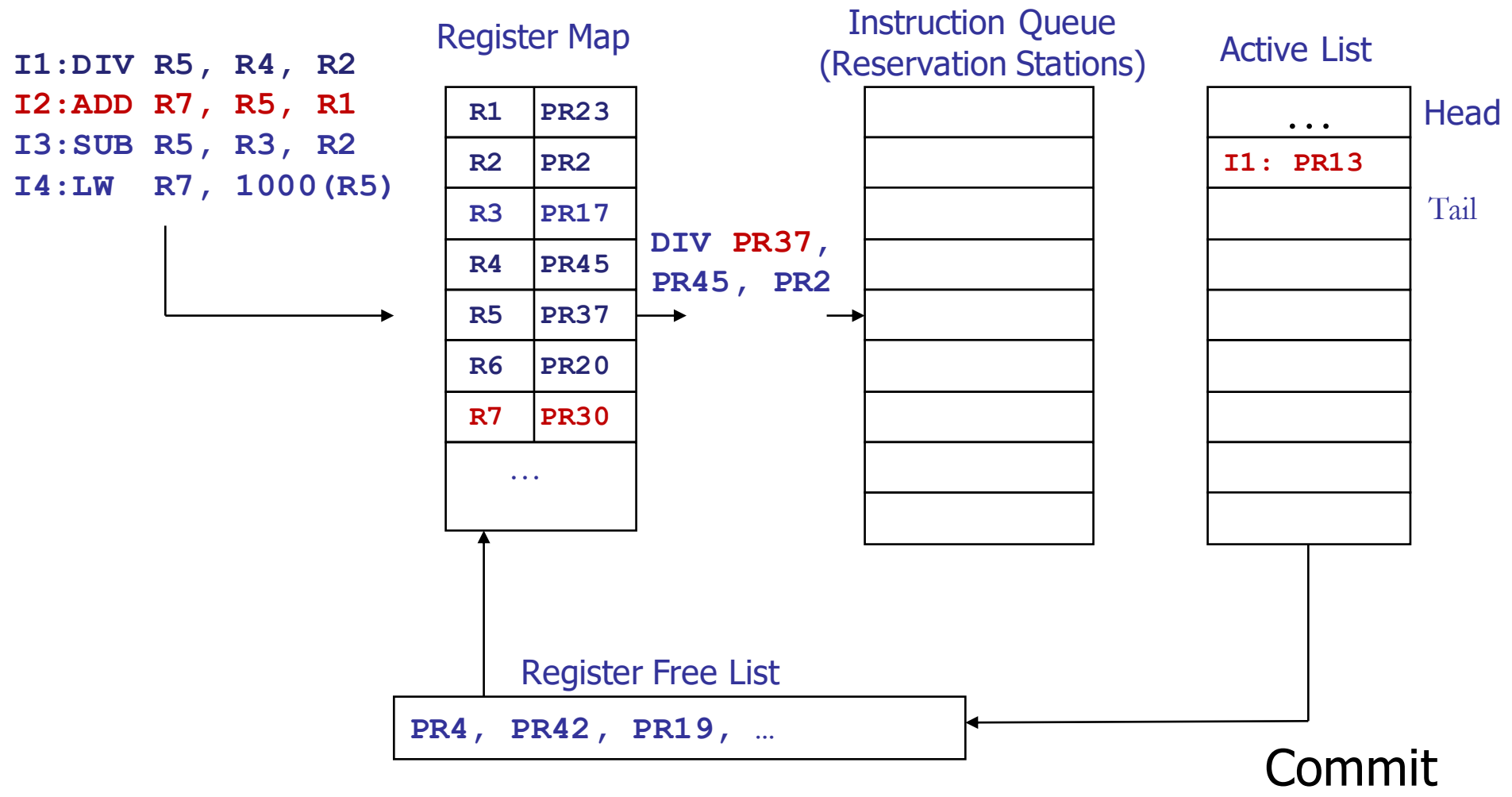
Register Free List

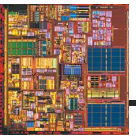
PR37, PR4, PR42, PR19, ...

Commit

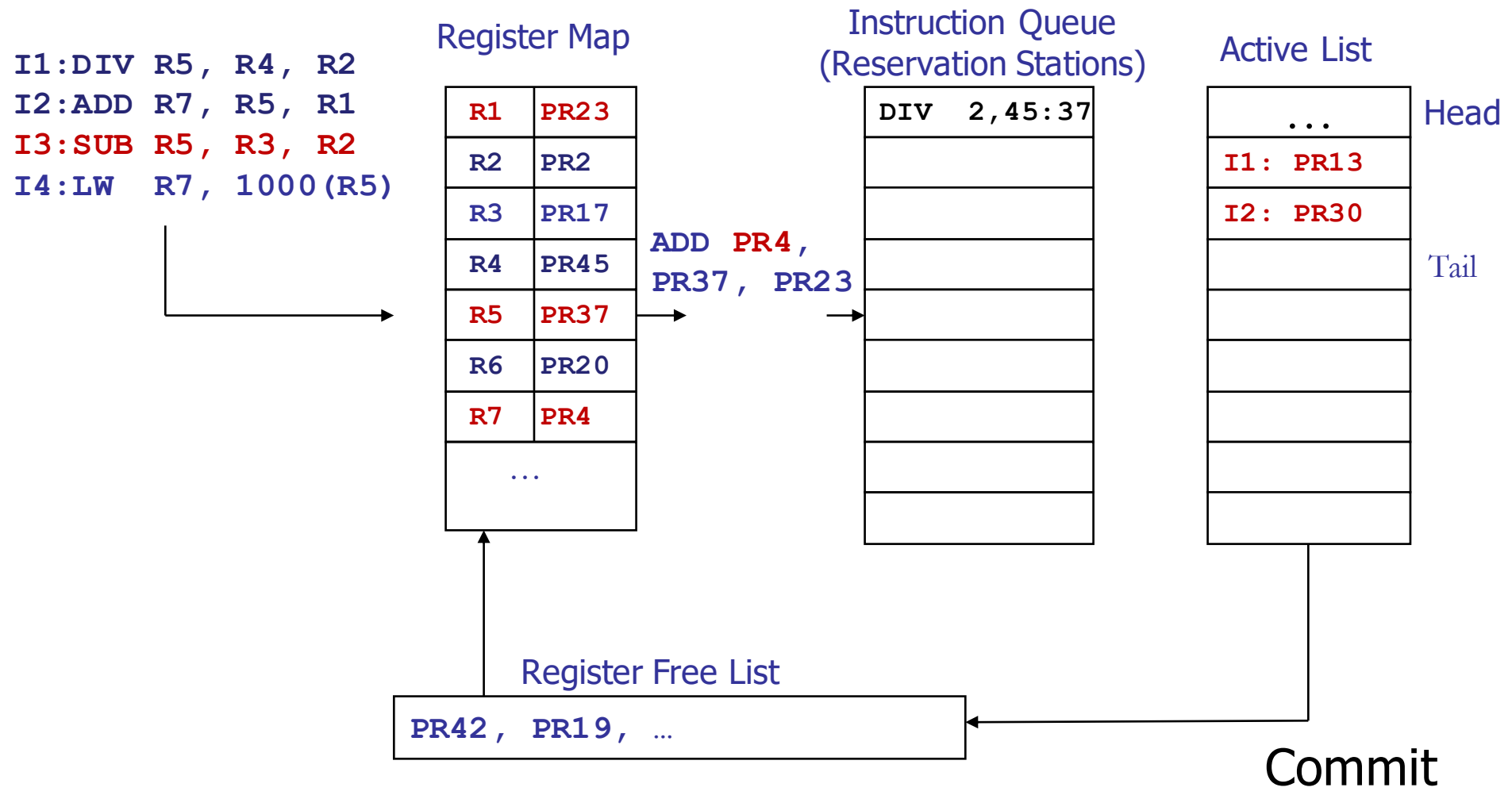


MIPS R10000, Recycling Physical Registers

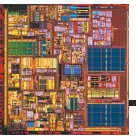


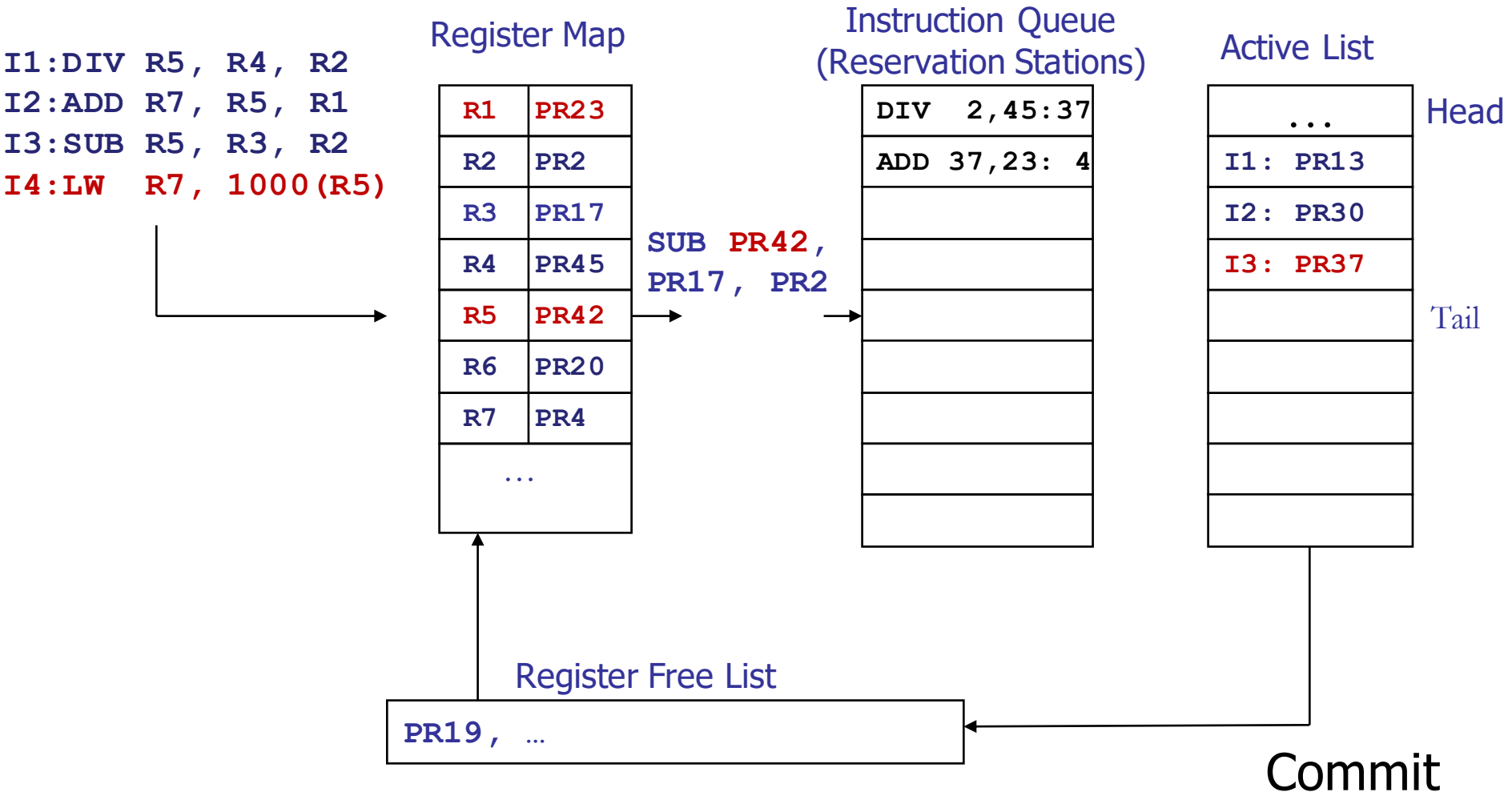


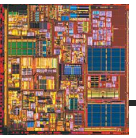
MIPS R10000, Recycling Physical Registers



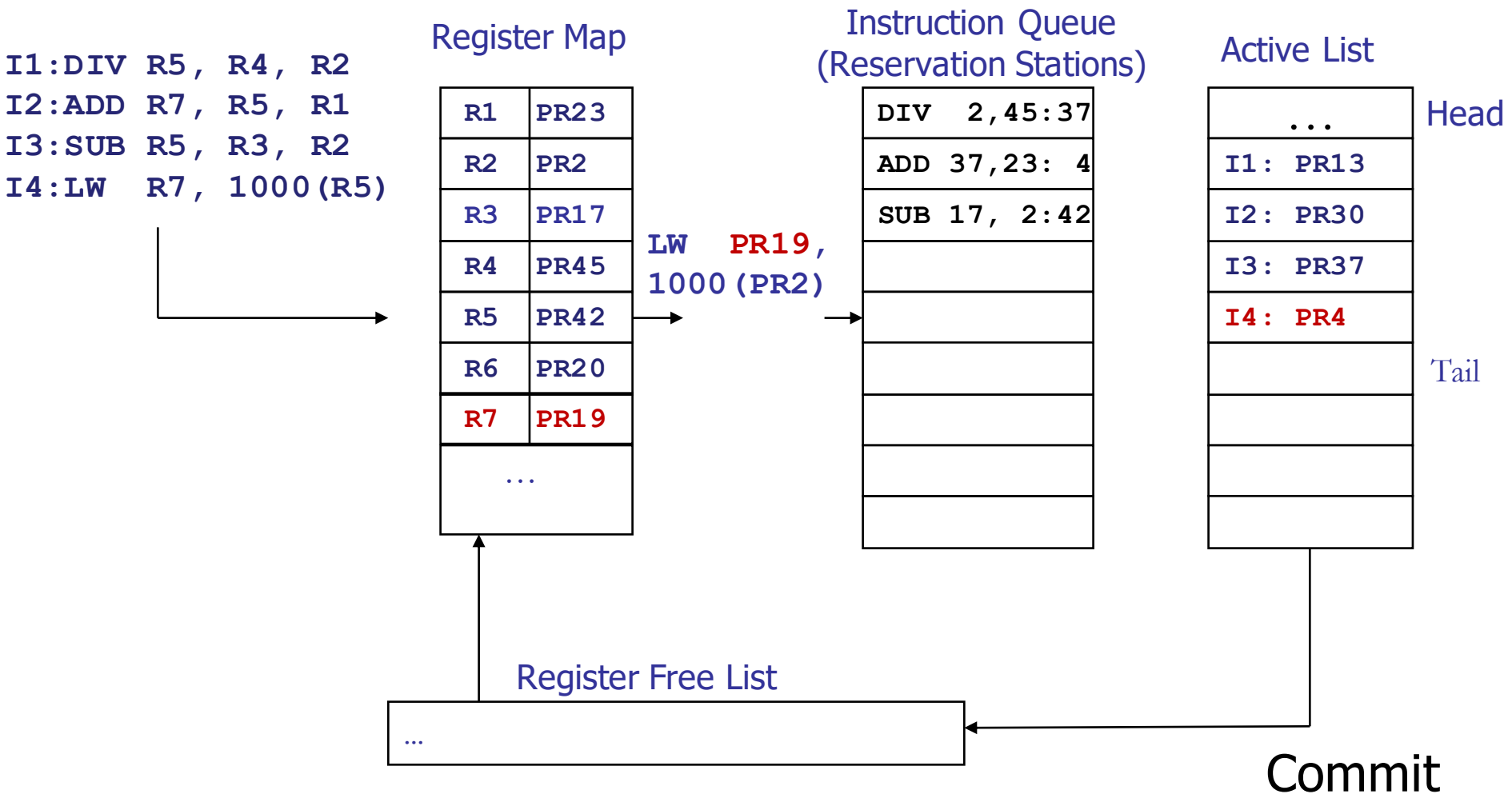
MIPS R10000, Recycling Physical Registers

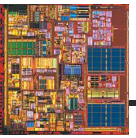




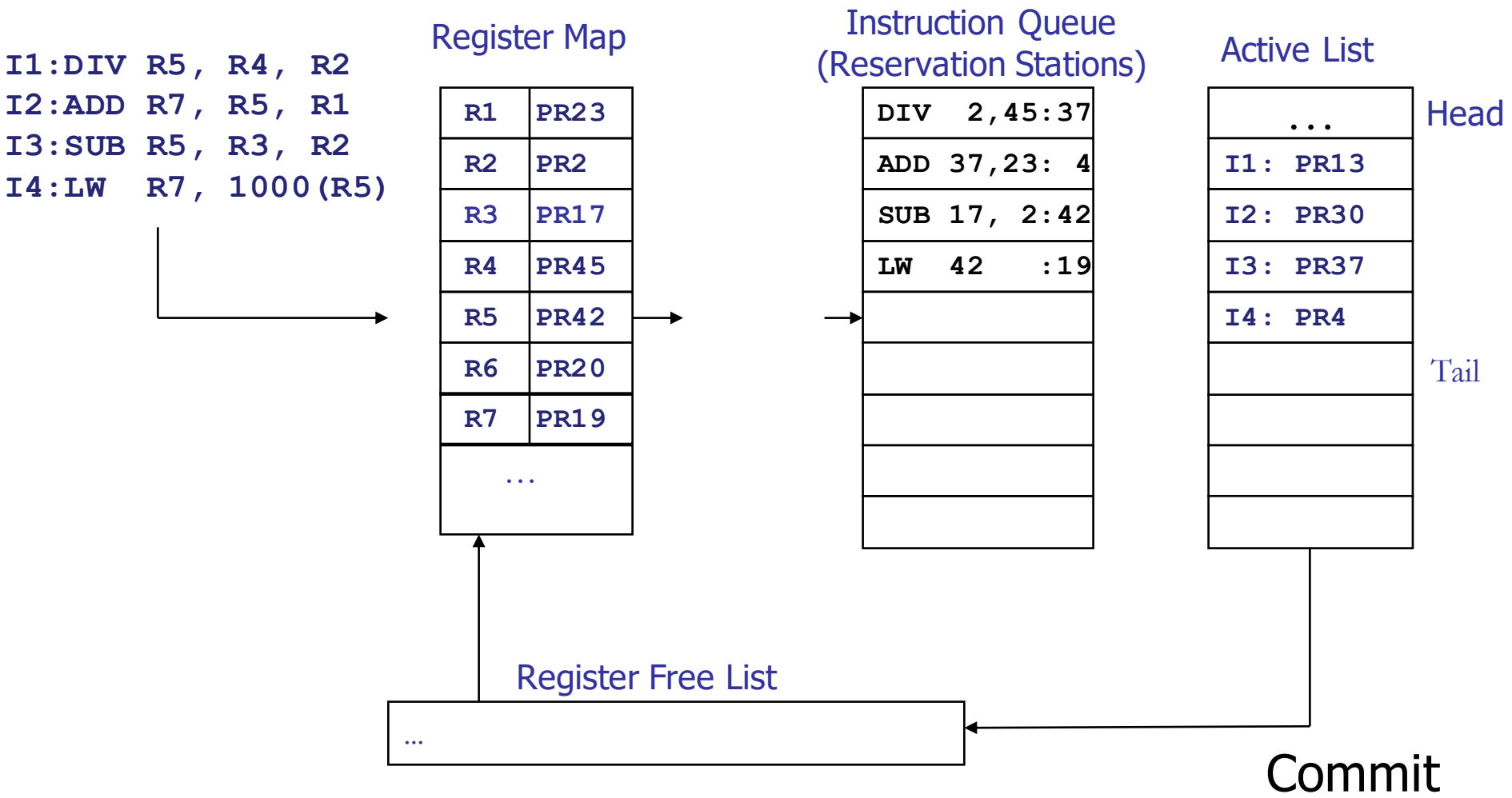


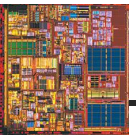
MIPS R10000, Recycling Physical Registers



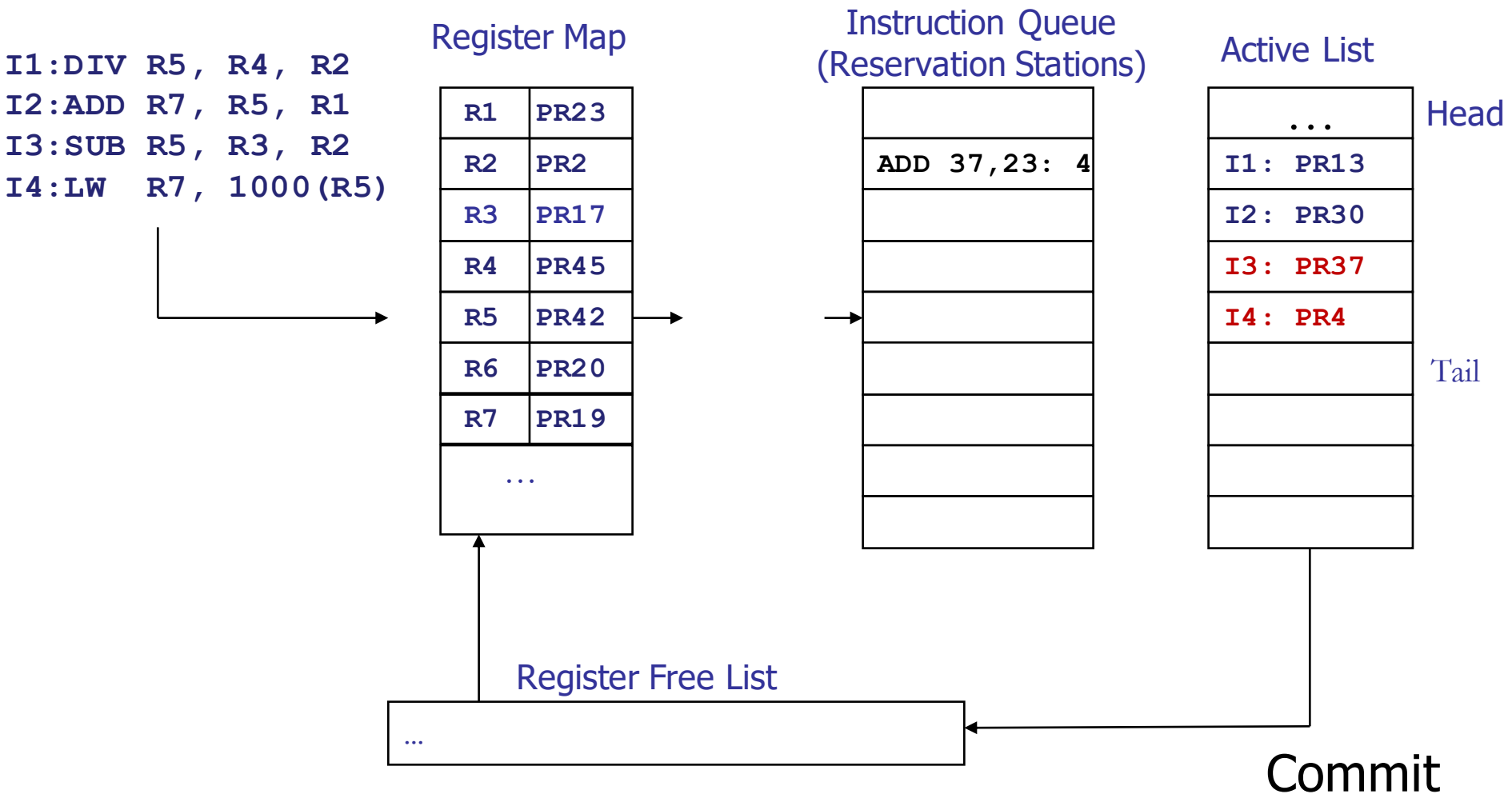


MIPS R10000, Recycling Physical Registers

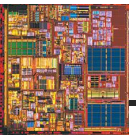




MIPS R10000, Recycling Physical Registers



I4, producing register 19, completes, broadcasts a completion signal to IQ



MIPS R10000, Recycling Physical Registers

```
I1:DIV R5, R4, R2
I2:ADD R7, R5, R1
I3:SUB R5, R3, R2
I4:LW  R7, 1000(R5)
```

Register Map

R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR42
R6	PR20
R7	PR19
...	

Instruction Queue (Reservation Stations)

ADD 37,23: 4

Active List

...
I1: PR13
I2: PR30
I3: PR37
I4: PR4

Head

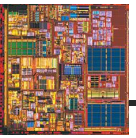
Tail

Register Free List

...

Commit

I1, producing register 37, completes, broadcasts a completion signal to IQ



MIPS R10000, Recycling Physical Registers

```
I1:DIV R5, R4, R2
I2:ADD R7, R5, R1
I3:SUB R5, R3, R2
I4:LW  R7, 1000(R5)
```

Register Map

R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR42
R6	PR20
R7	PR19
...	

Instruction Queue (Reservation Stations)

ADD 37,23: 4

Active List

...
~~I1: PR13~~
I2: PR30
I3: PR37
I4: PR4

Head

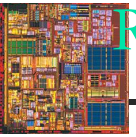
Tail

Register Free List

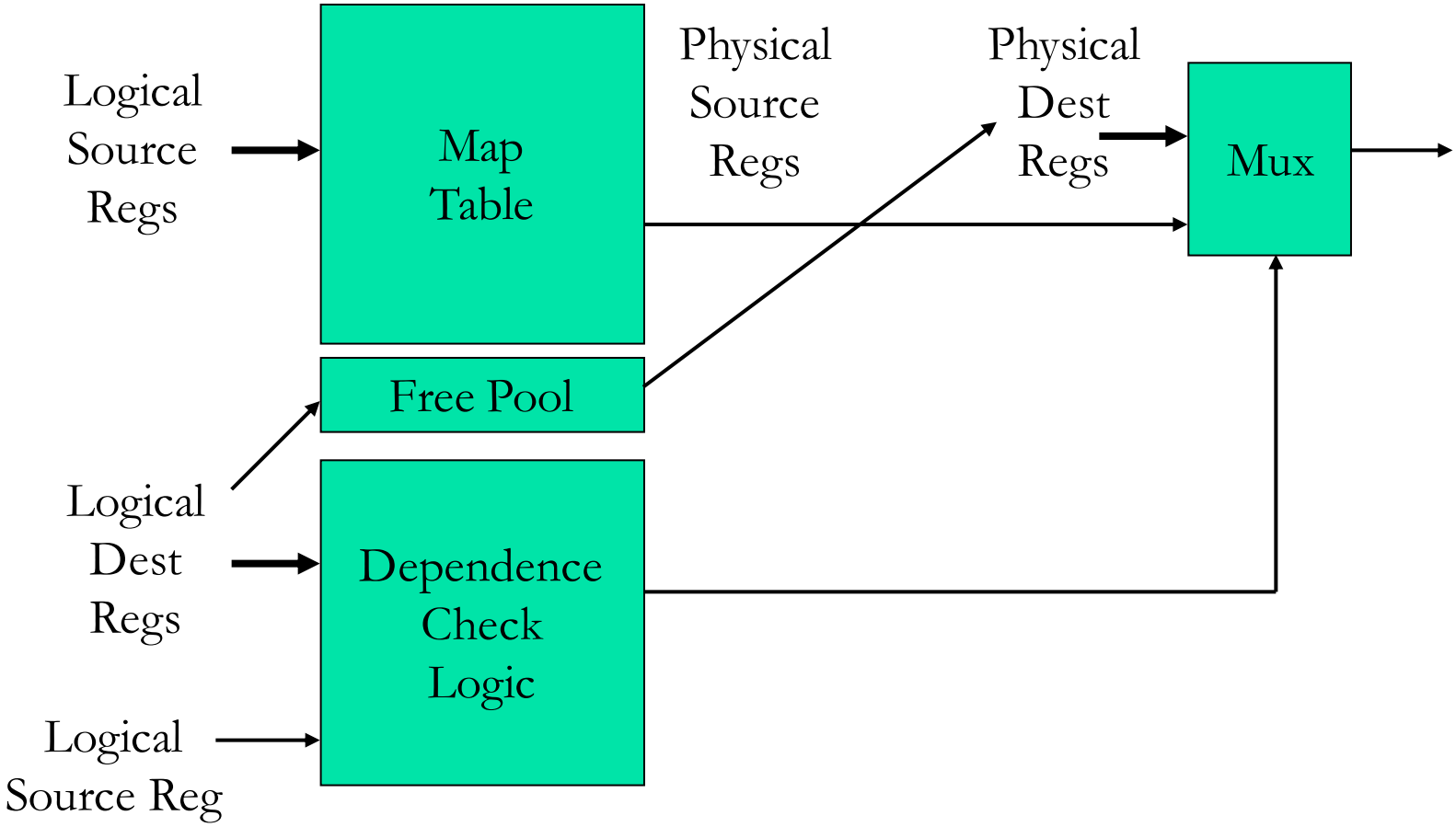
..., PR13

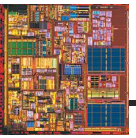
Commit

I2, producing register 4, completes, broadcasts a completion signal to IQ
I1, commits



Register Rename Logic

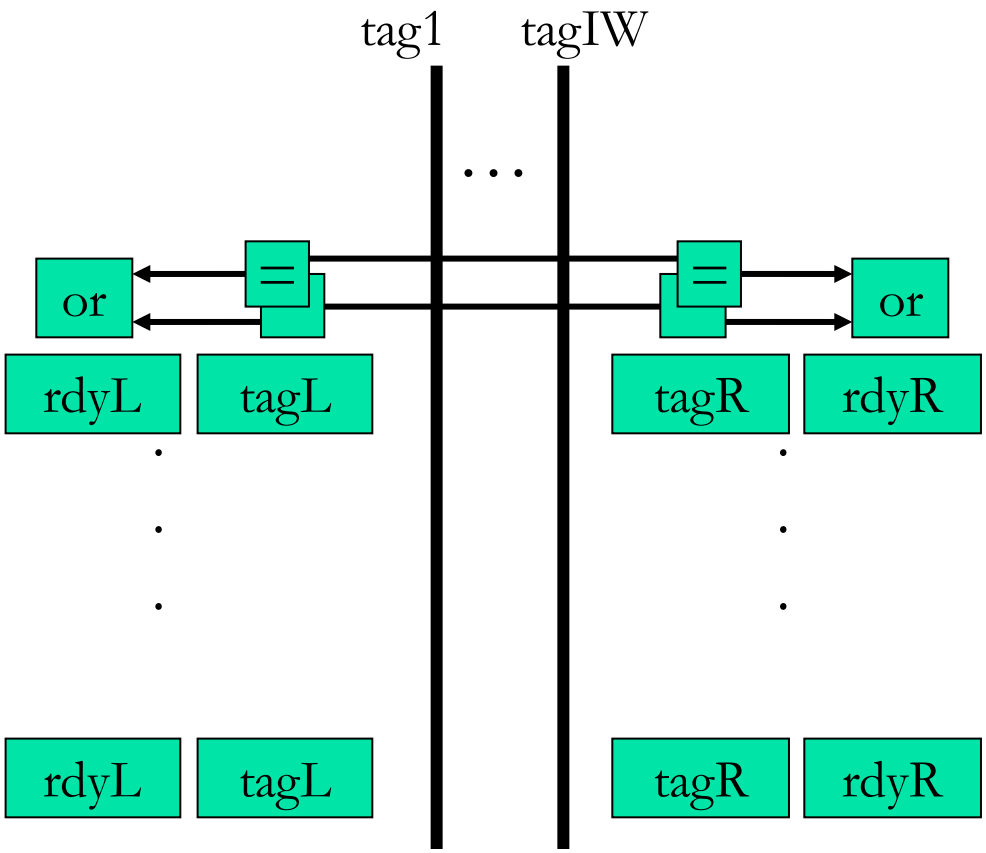


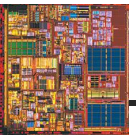


Instr Scheduling: Wakeup & Select

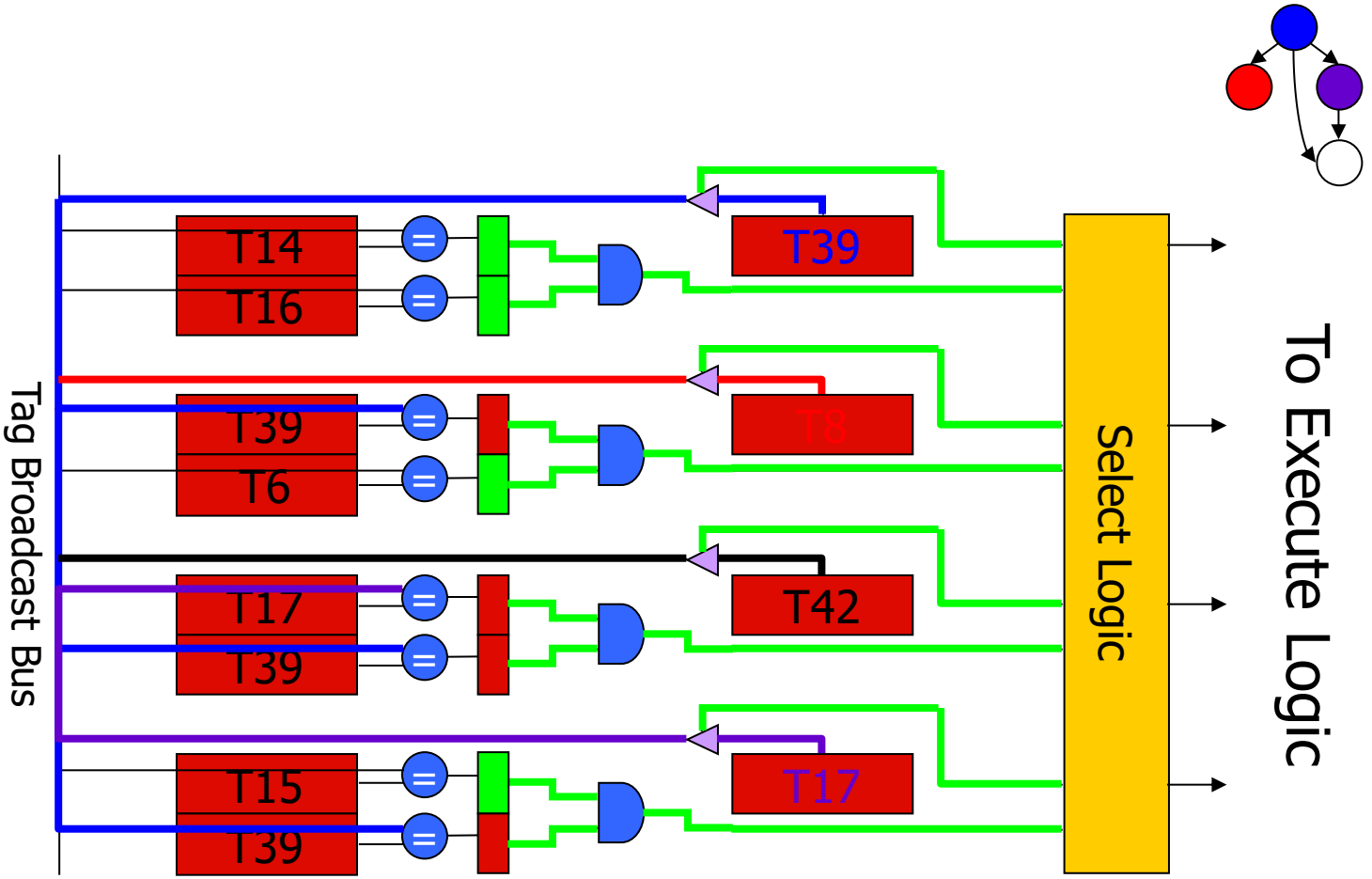
- Wakeup Logic
  - To notify the resolution of data dependency of input operands
  - Wake up instructions with zero input dependency
- Select Logic
  - Choose and fire ready instructions
  - Deal with structure hazard
- Wakeup-select is likely on the critical path
  - Associative match

Wakeup Logic

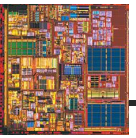




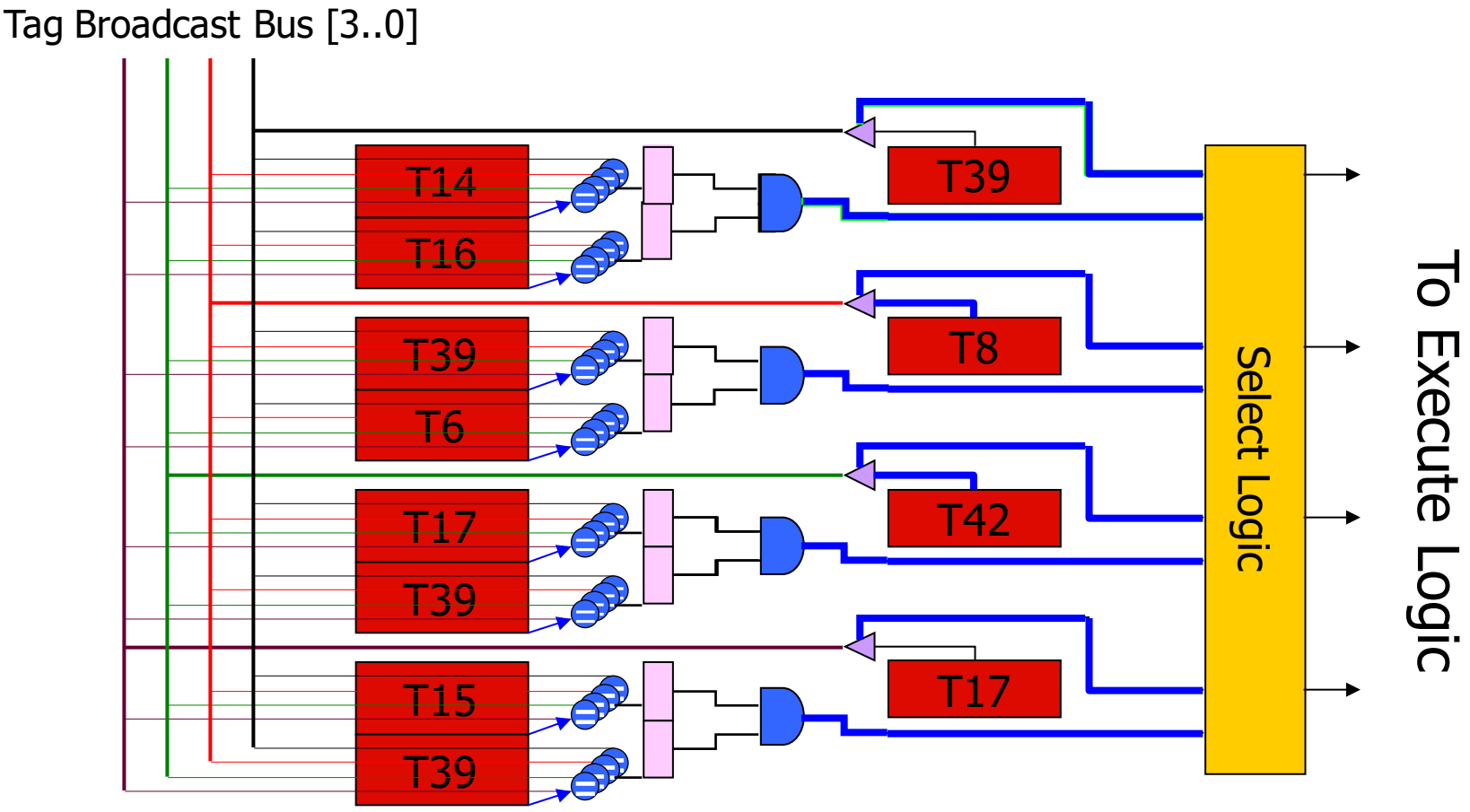
Scalar Scheduler (Issue Width = 1)



From Prof. G. Loh's Slide



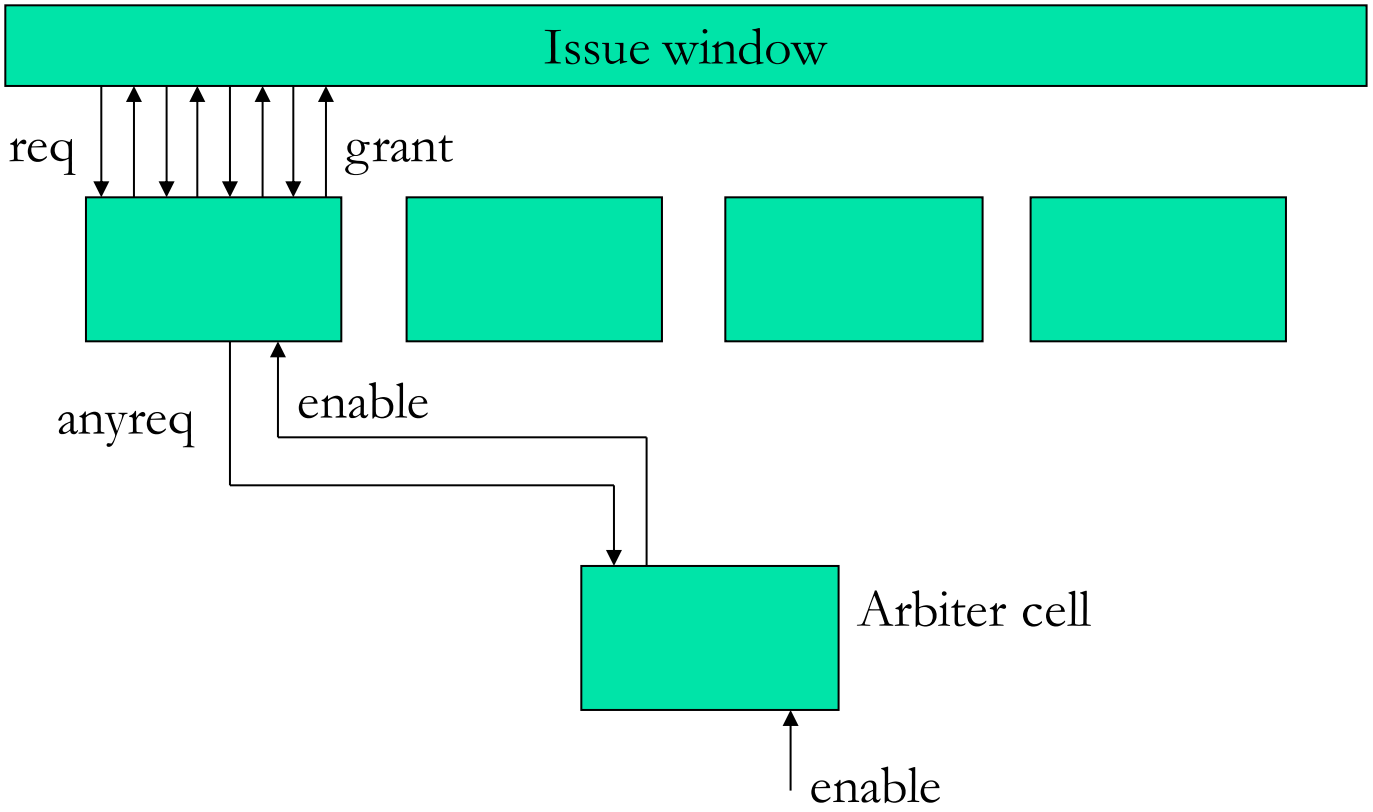
Superscalar Scheduler (Width = 4)



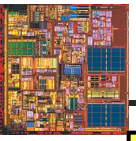
Snapshot of RS (only 4 entries shown)

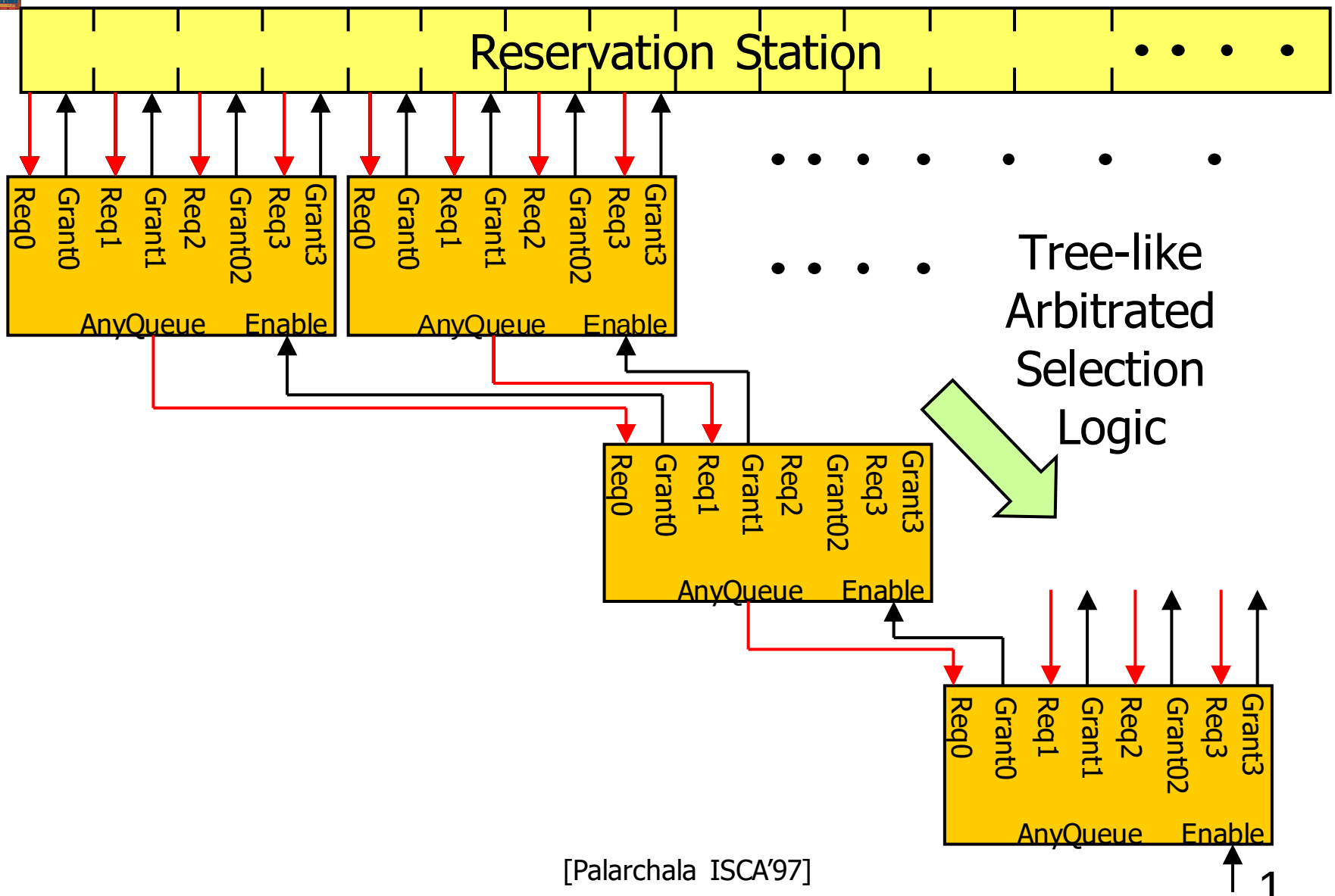
Adapted from Prof. G. Loh's Slide

Selection Logic



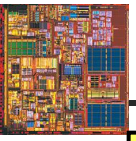
Simple (?) Select Logic Implementation

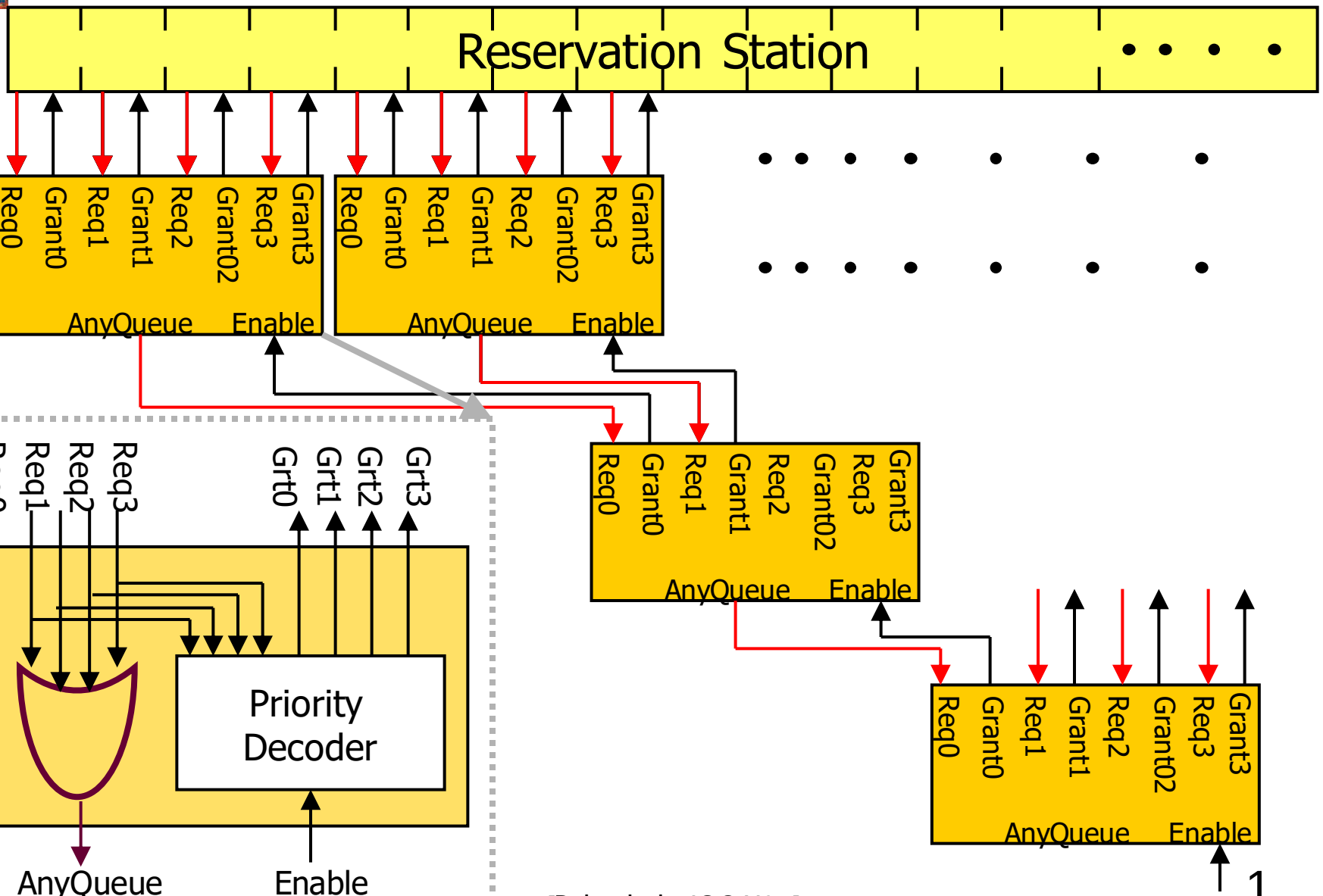




[Palarchala ISCA'97]

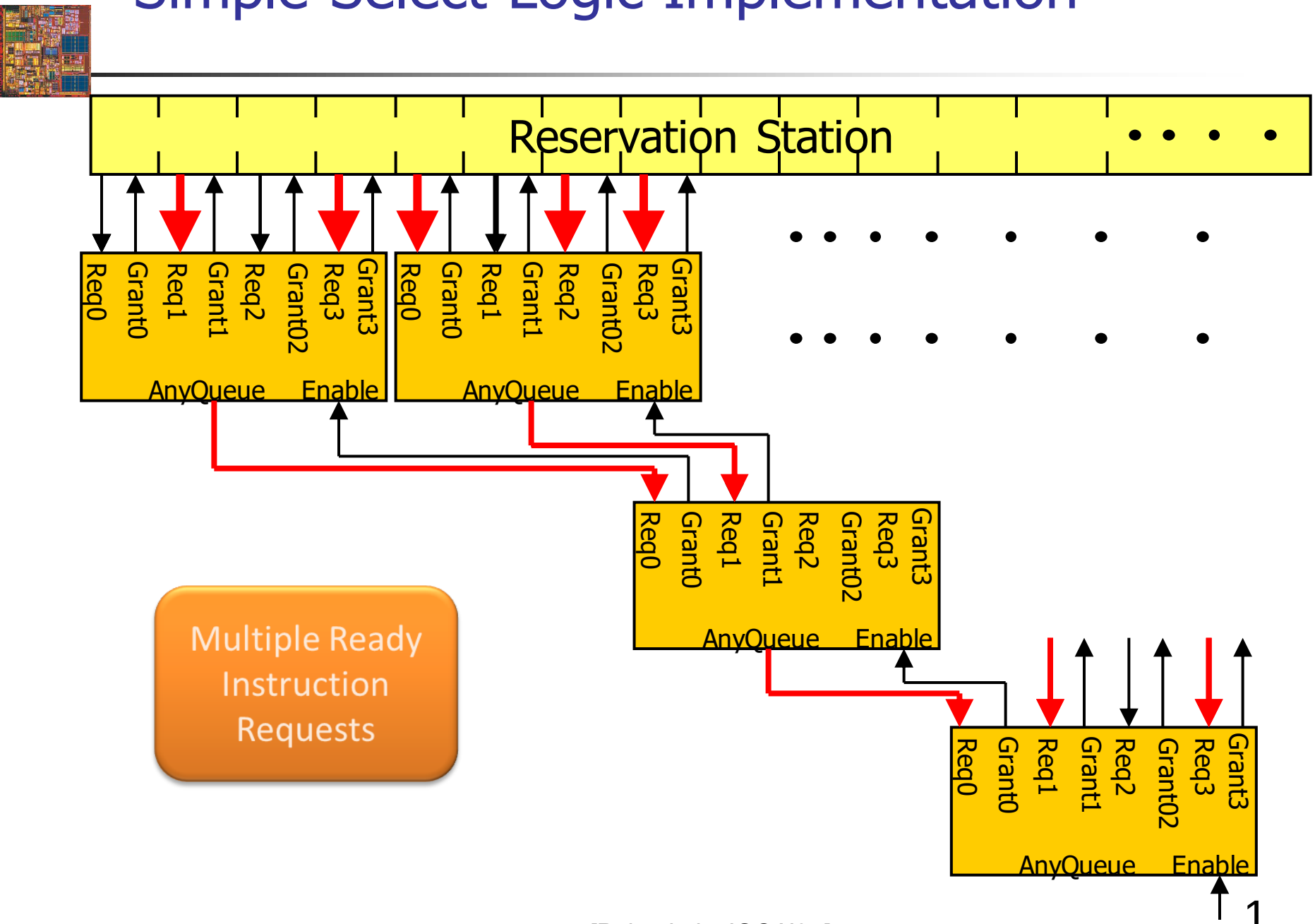
Simple Select Logic Implementation





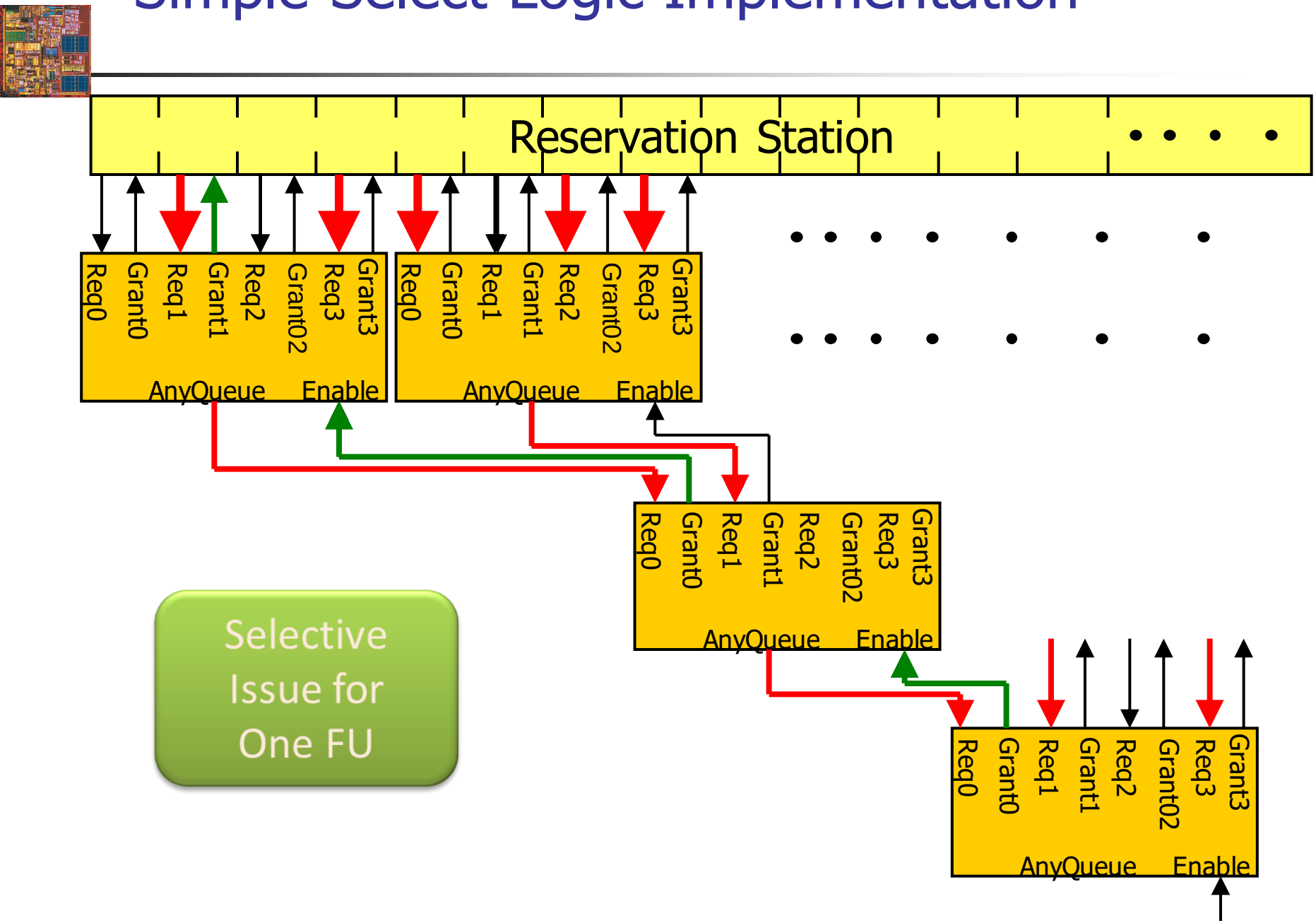
[Palarchala ISCA'97]

Simple Select Logic Implementation



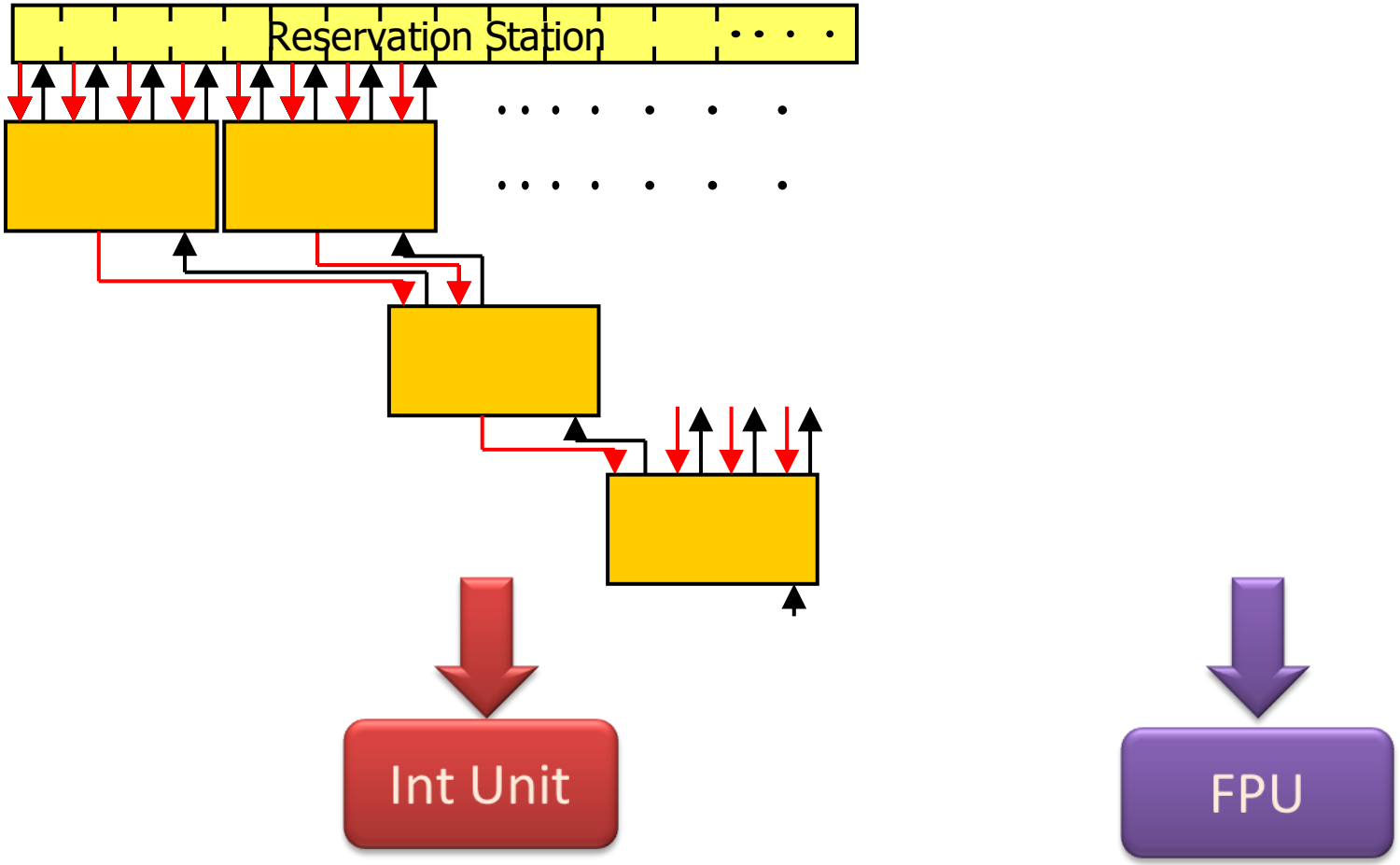
[Palarchala ISCA'97]

Simple Select Logic Implementation



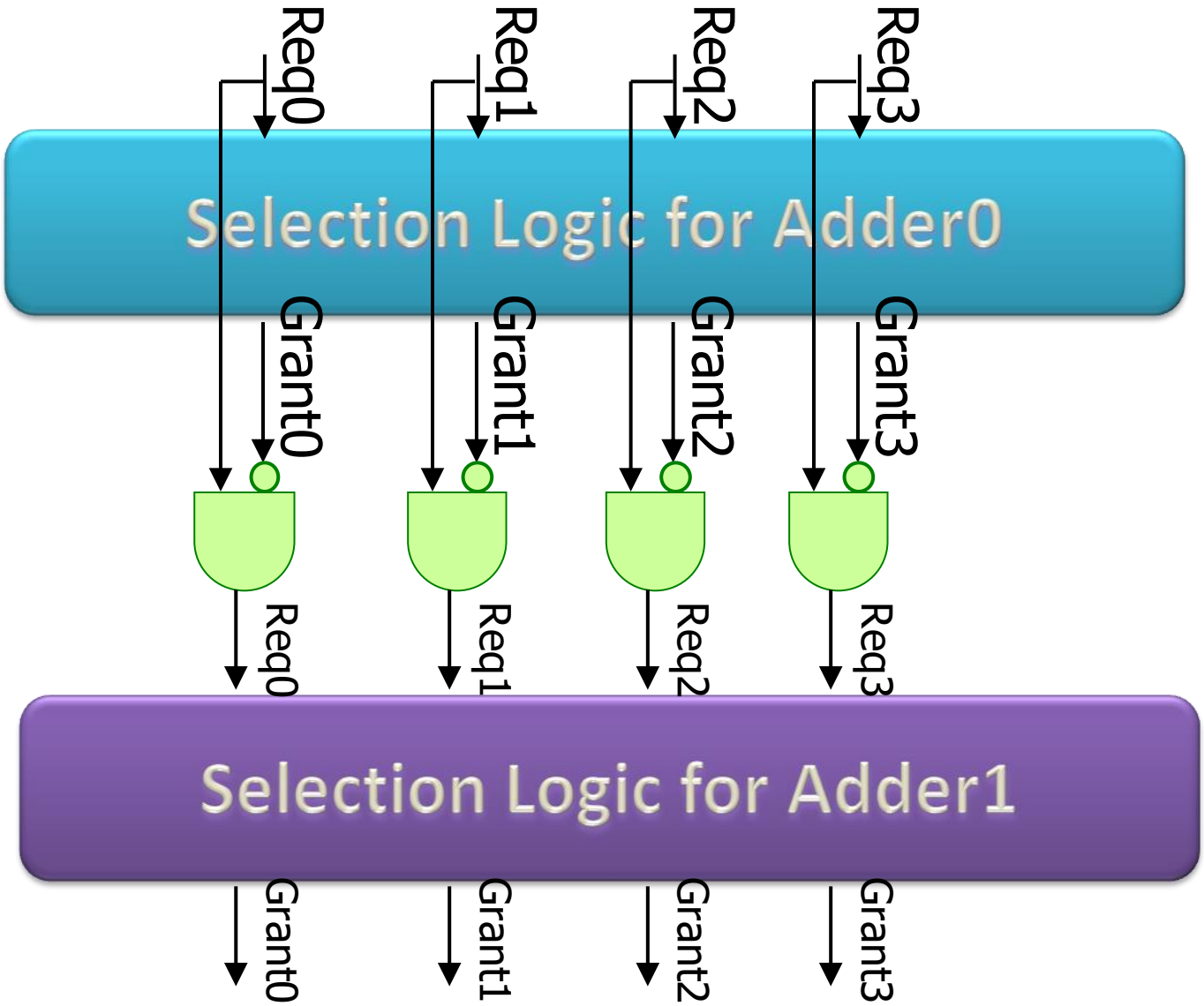
Issues to Distinctive Functional Units

istributed Instruction Windows (e.g., MIPS R1000 or Alpha 21264)

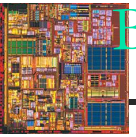


Faster to have separate instruction schedulers for different instruction

Dual Issues to Multiple Units



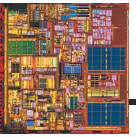
[Palarchala Dissertation]



Bypass Delay

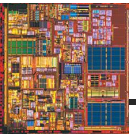
- The number of bypass paths equals $2xIW^2xS$ (S is the number of pipeline stages)
- Wire length ~ IW, hence, delay ~ IW^2
- The layout and pipeline depth (capacitive load) also matter

Other structures



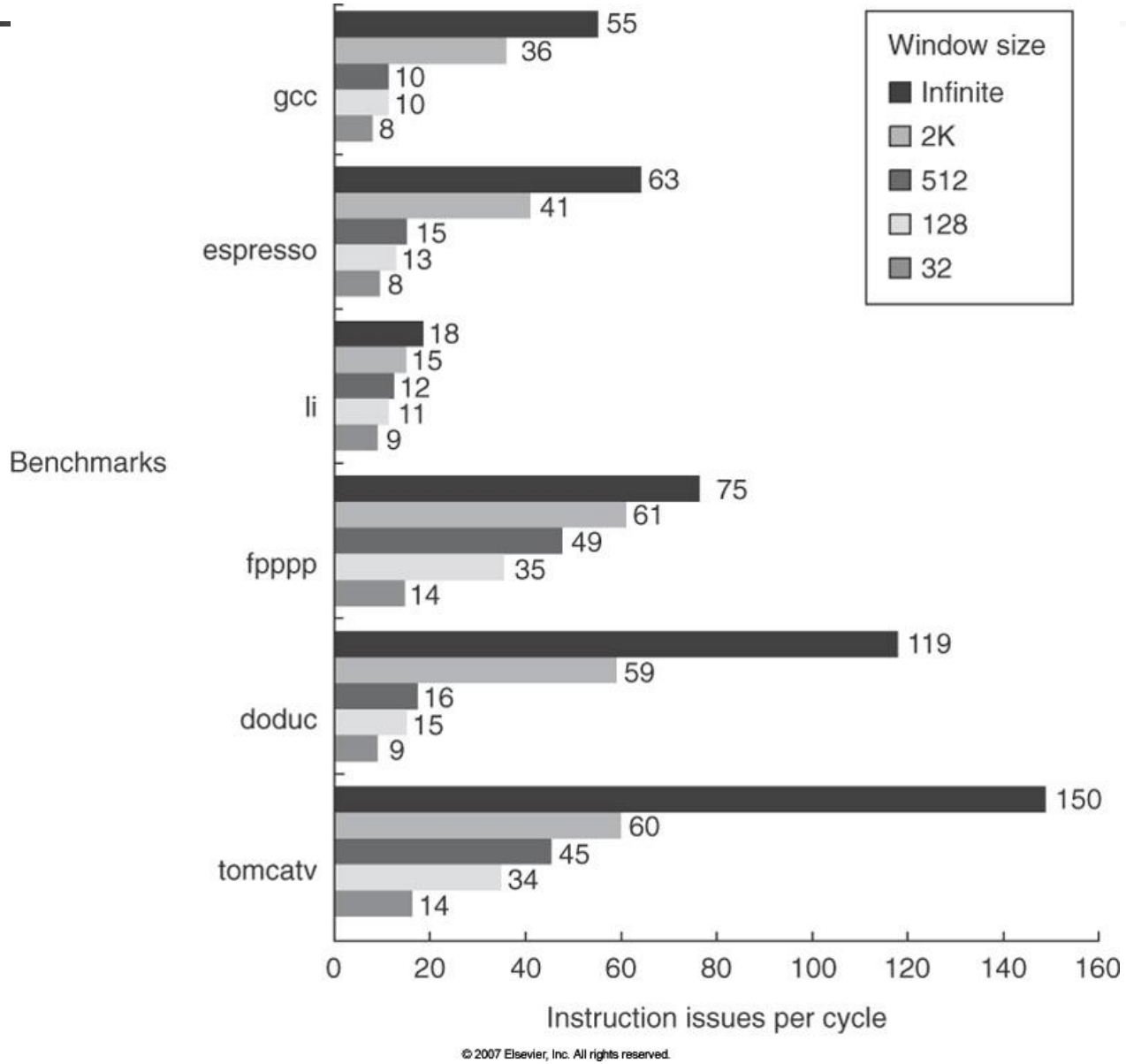
- Registers
- Caches

Limitations of ILP

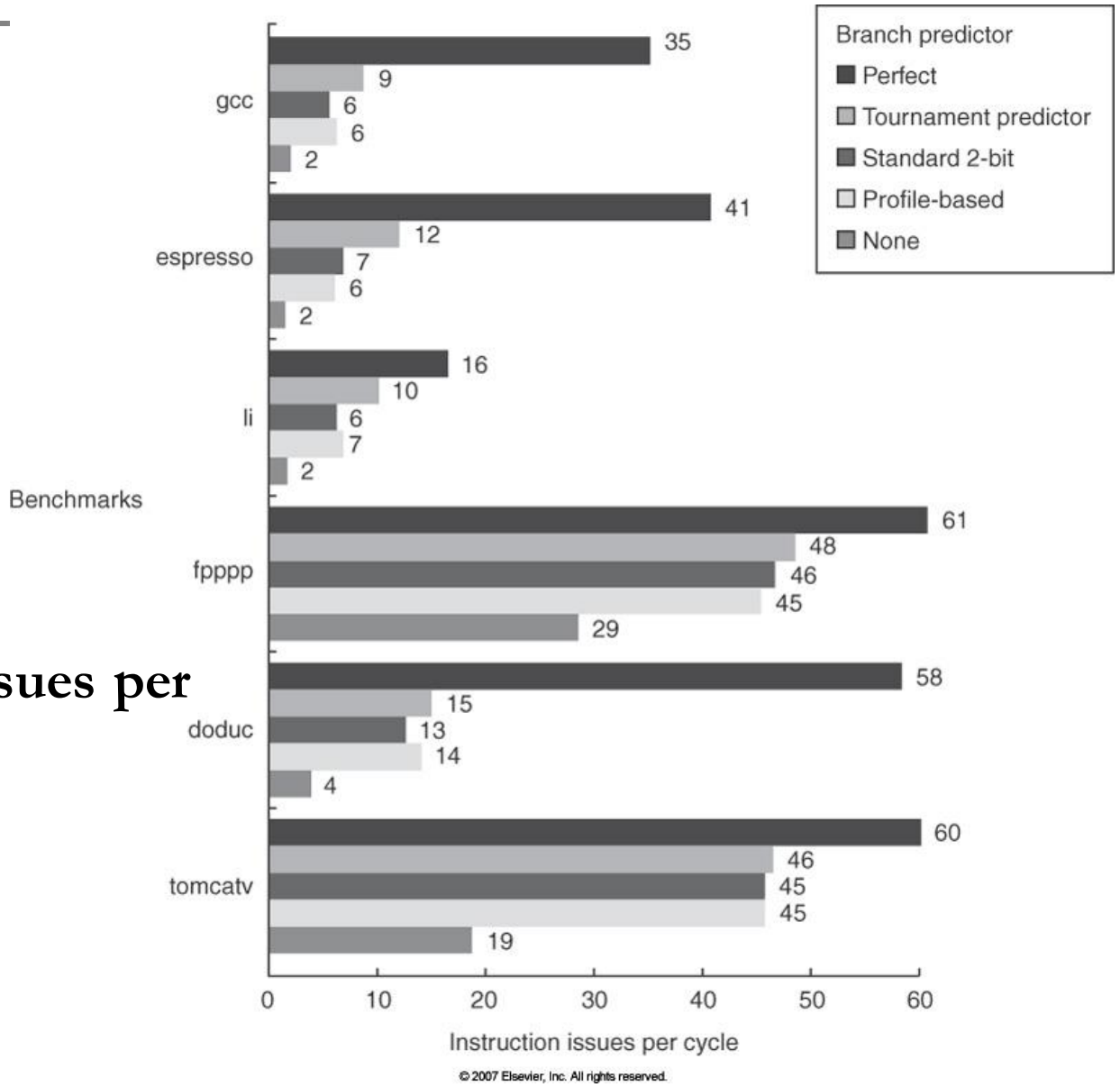


- Assume an ideal processor model
  - Register renaming
    - Infinite number of virtual registers
    - All WAW and WAR hazards can be removed
  - Branch prediction
    - Perfect branch prediction
  - Jump prediction
    - All jumps are perfectly predicted
  - Memory address alias analysis
    - All memory addresses are known exactly
  - Perfect caches
    - All memory accesses take 1 clock cycle

Impact of Window Size



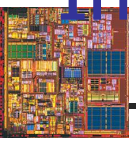
Impact of Branch Prediction

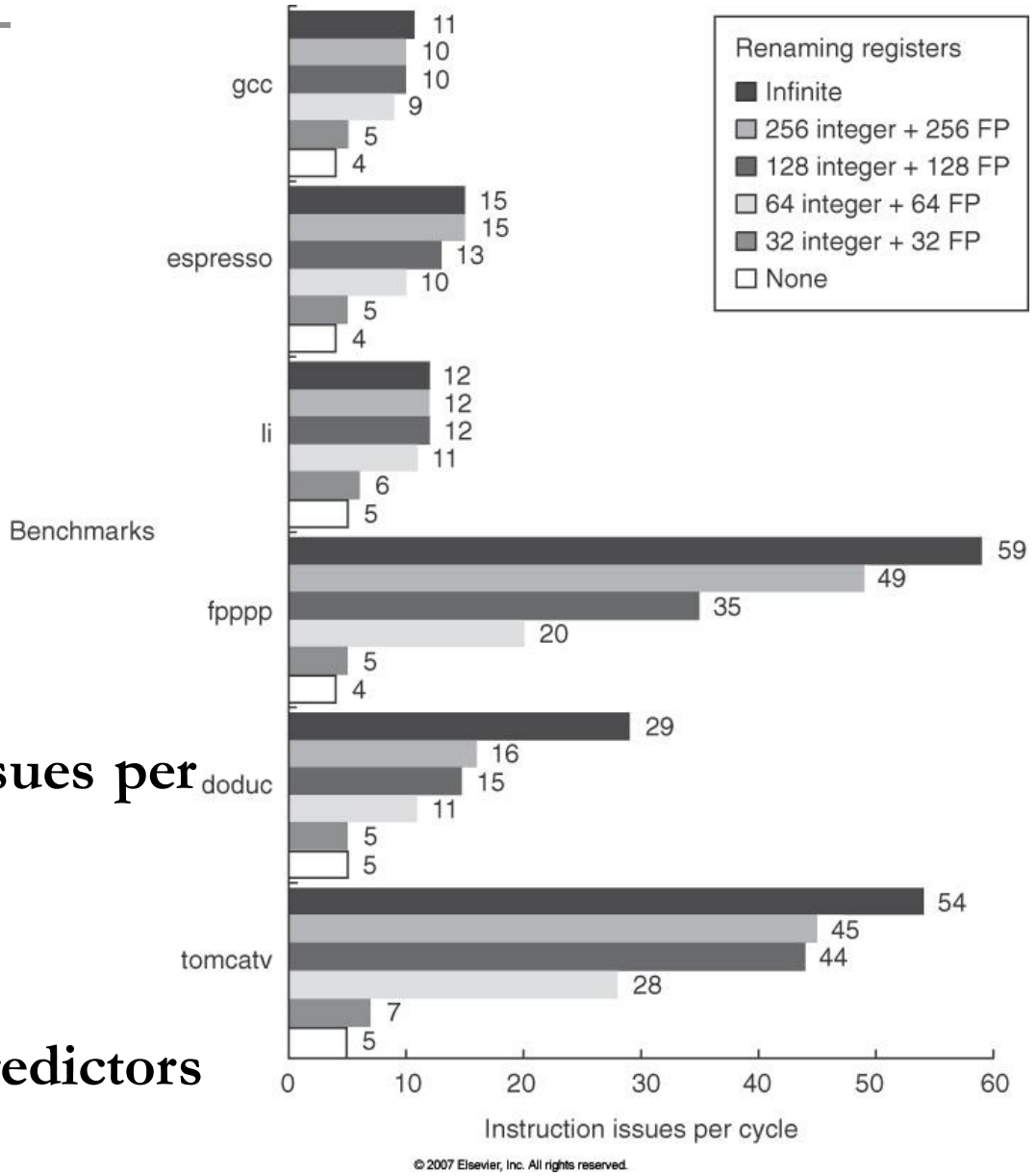


2K window

Max. 64 instruction issues per cycle

Impact of Finite Registers





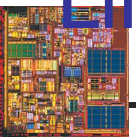
2K window

Max. 64 instruction issues per cycle

8K entry tournament predictors

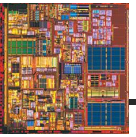
2K jump and return predictors

Limits of Multiple-Issue Processors: Revisited



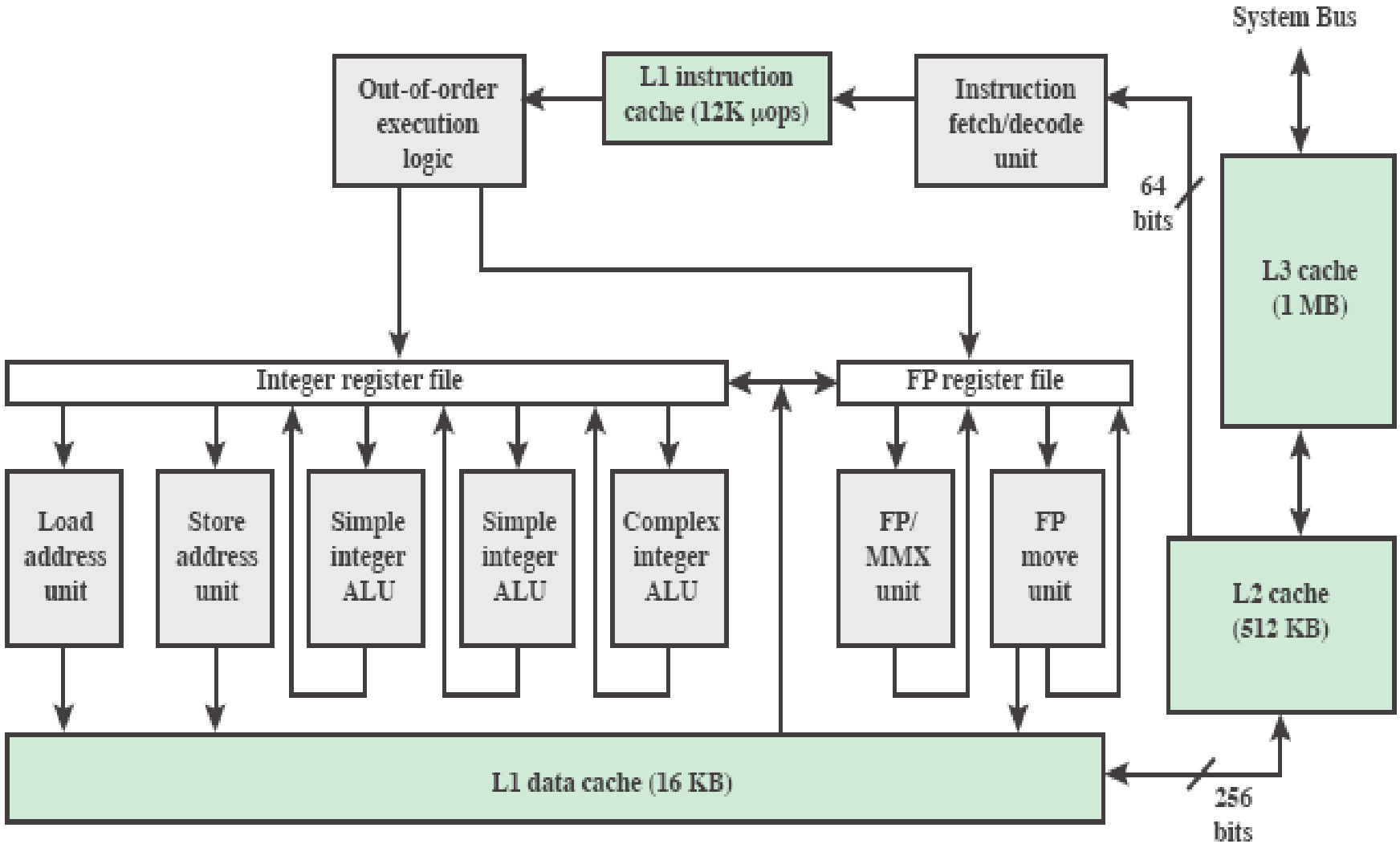
- Doubling the issue rate above the current 3-6 issue, i.e. 6-12 issue requires
  - Issue 3-4 data memory accesses per cycle
  - Resolve two or three branches per cycle
  - Rename and access more than 20 registers per cycle
  - Fetch 12-24 instructions per cycle
  - ➔ The complexity of implementing these capabilities would sacrifice the maximum clock rate
- Another issue is power!
  - Modern microprocessors are primarily power limited.
    - *Static power grows proportionally to the transistor count*
    - *Dynamic power is proportional to the product of the number of switching transistors and the switching rate*
  - Microprocessors trying to achieve both a high IPC and a

Limits of Multiple-Issue Processors

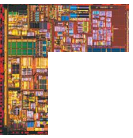


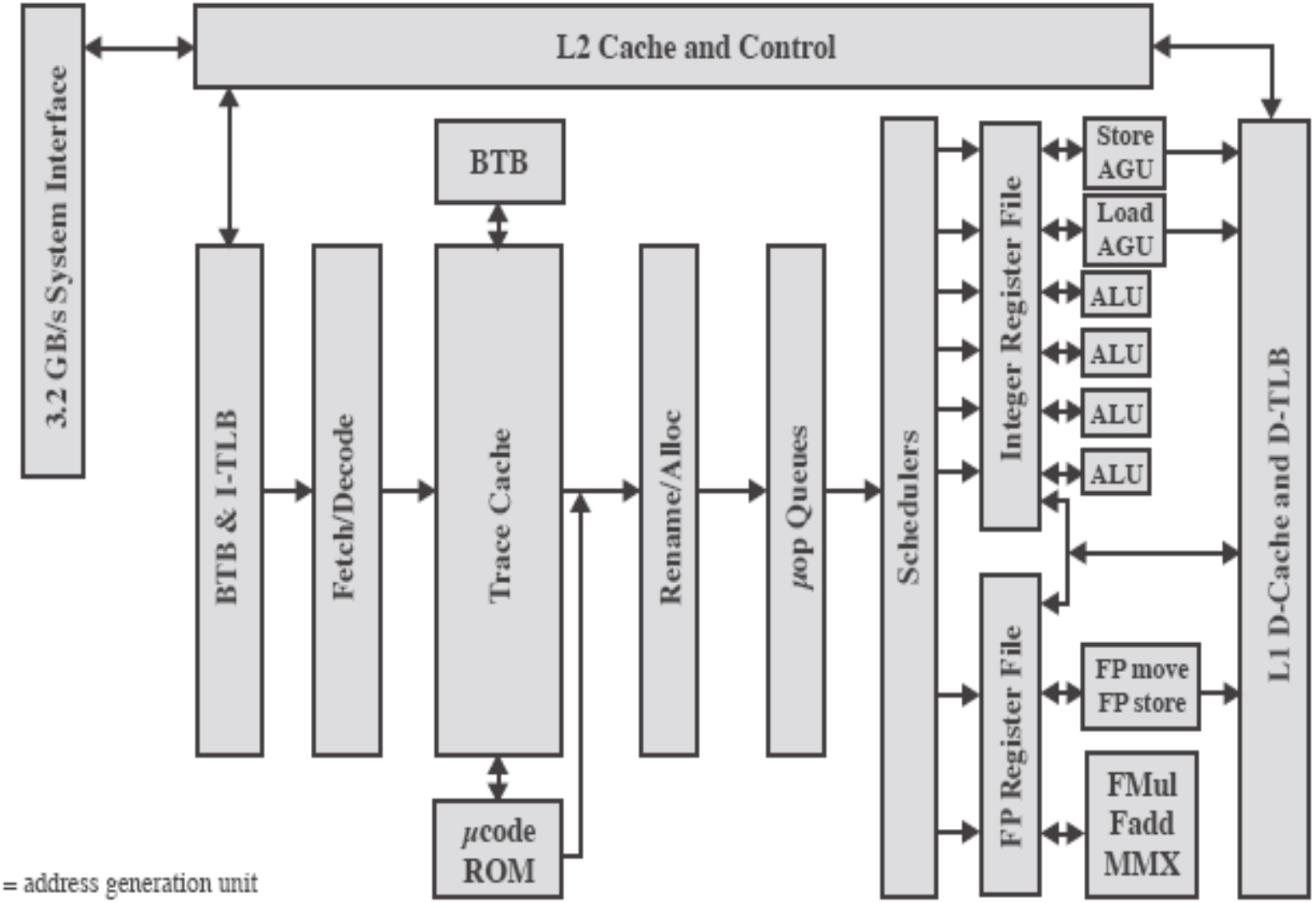
- Energy inefficiency of multiple-issue processors
  - Multiple instruction issue incurs overhead in logic that grows faster than the issue rate increase
    - Dependence checking, register renaming, wakeup and select, etc.
    - Without voltage reduction, higher IPC will lead to lower performance/watt!
  - Growing gap between peak issue rate and sustained performance
    - The number of transistor switching is proportional to the peak issue rate
    - The performance is proportional to the sustained rate
    - Thus, growing gap translates to increasing energy per unit performance!
- For example, speculation is inherently inefficient
  - It can never be perfect, thus there is inherently waste in executing computations before we know that whether the path is taken or not

Example: Pentium 4
A Superscalar CISC Machine



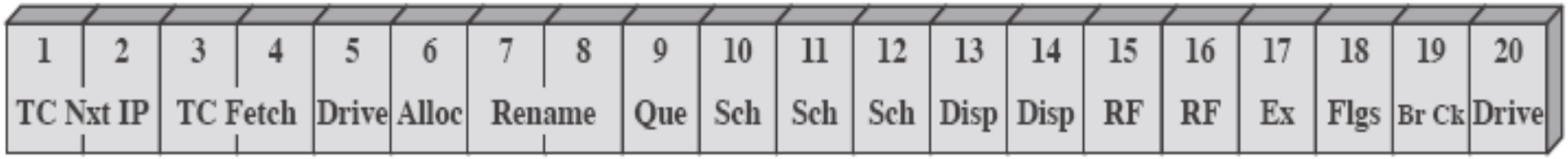
Pentium 4 alternate view





AGU = address generation unit
BTB = branch target buffer
D-TLB = data translation lookaside buffer
I-TLB = instruction translation lookaside buffer

Pentium 4 pipeline



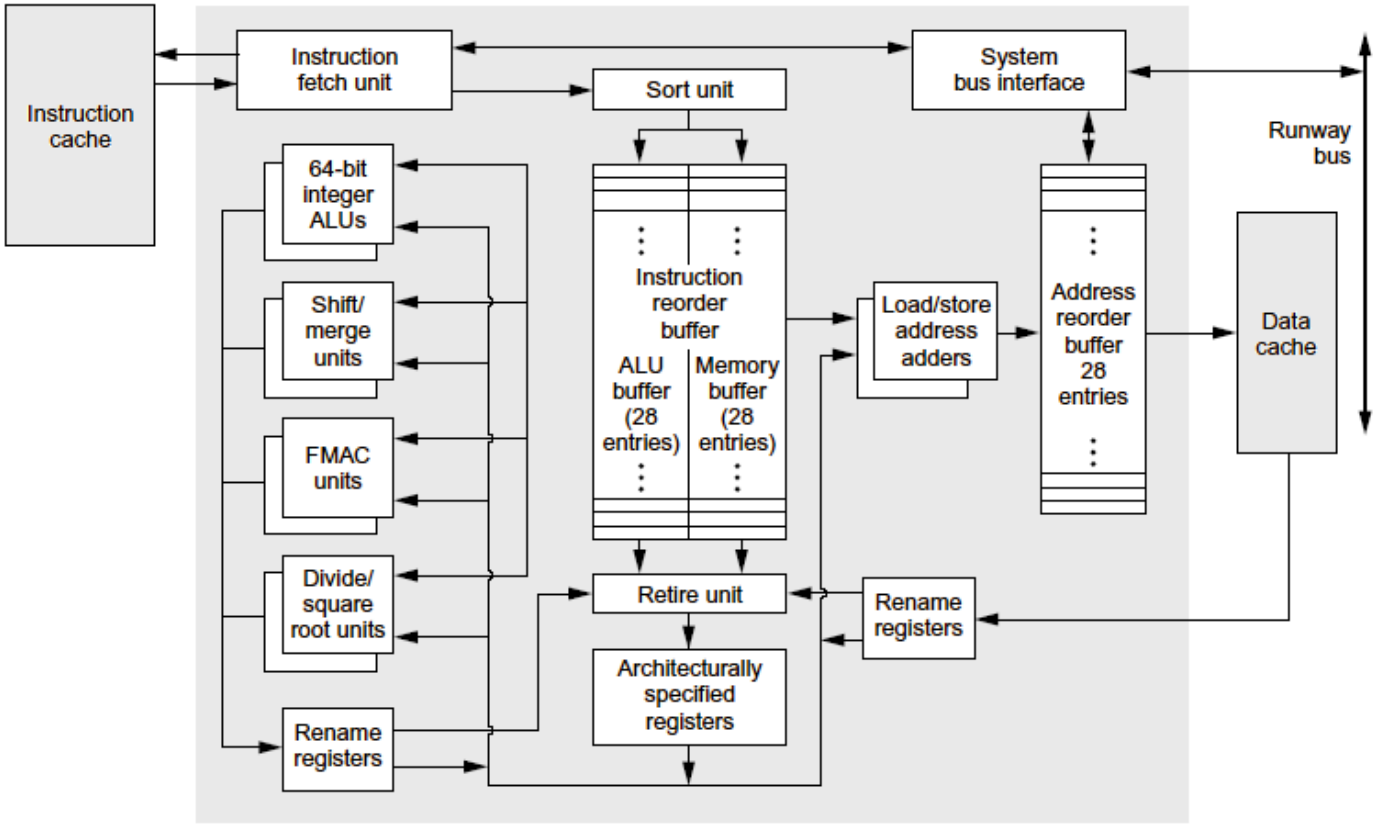
TC Next IP = trace cache next instruction pointer
TC Fetch = trace cache fetch
Alloc = allocate

Rename = register renaming
Que = micro-op queuing
Sch = micro-op scheduling
Disp = Dispatch

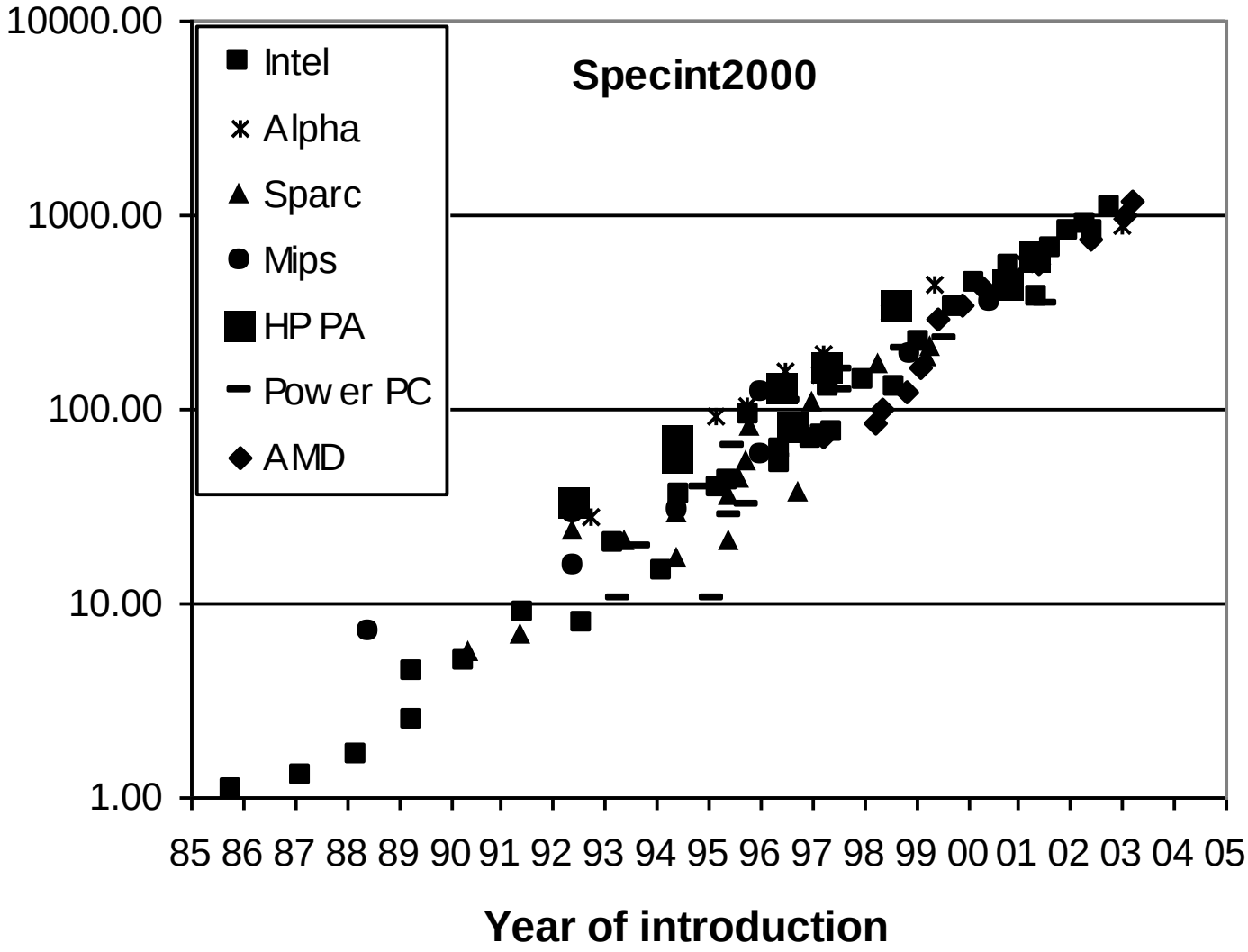
RF = register file
Ex = execute
Flgs = flags
Br Ck = branch check

20 stages !

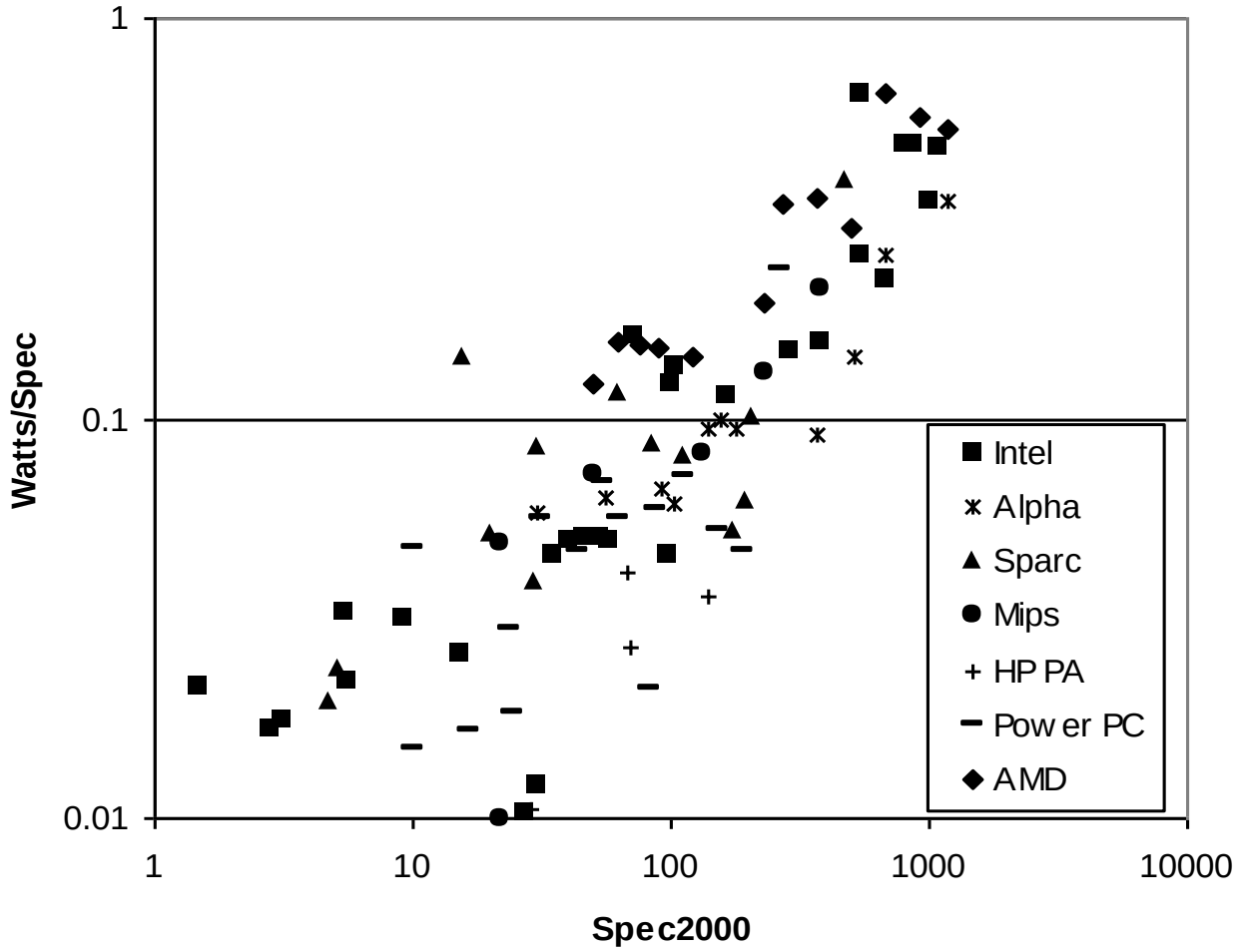
HP PA-RISC 8000



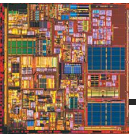
Processor Performance with Time



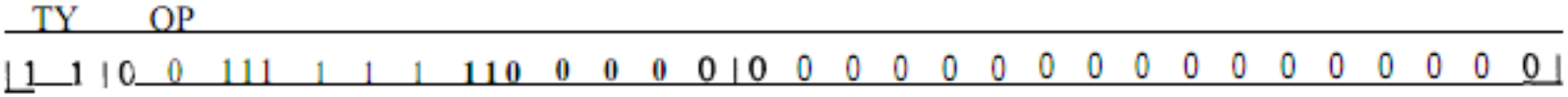
Price being paid



MIPS-X (1988)



4.58. **hsc - Halt and Spontaneously Combust**



Assembler

hsc

Operation

Reg(31) ⇐ PC
The processor stops fetching instructions and self destructs.

Note that the contents of Reg(31) are actually lost.

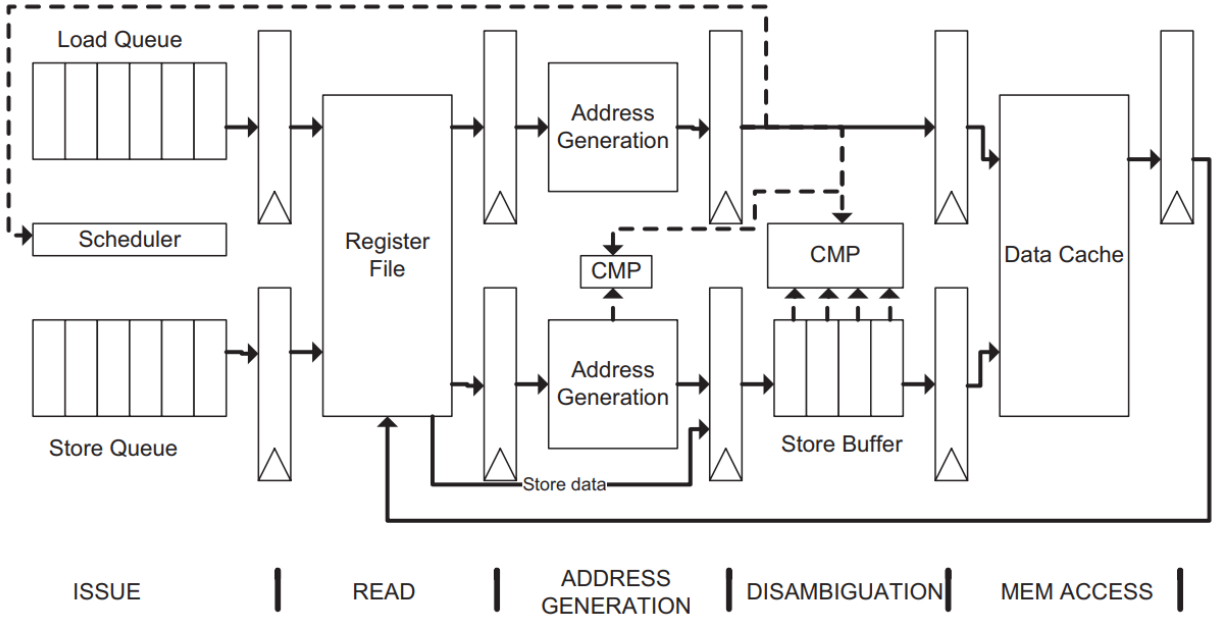
Description

This is executed by the processor when a protection violation is detected. It is a privileged instruction available only on the *-NSA* versions of the processor.

ftp://reports.stanford.edu/pub/cstr/reports/csl/tr/86/289/CSL-TR-86-289.pdf

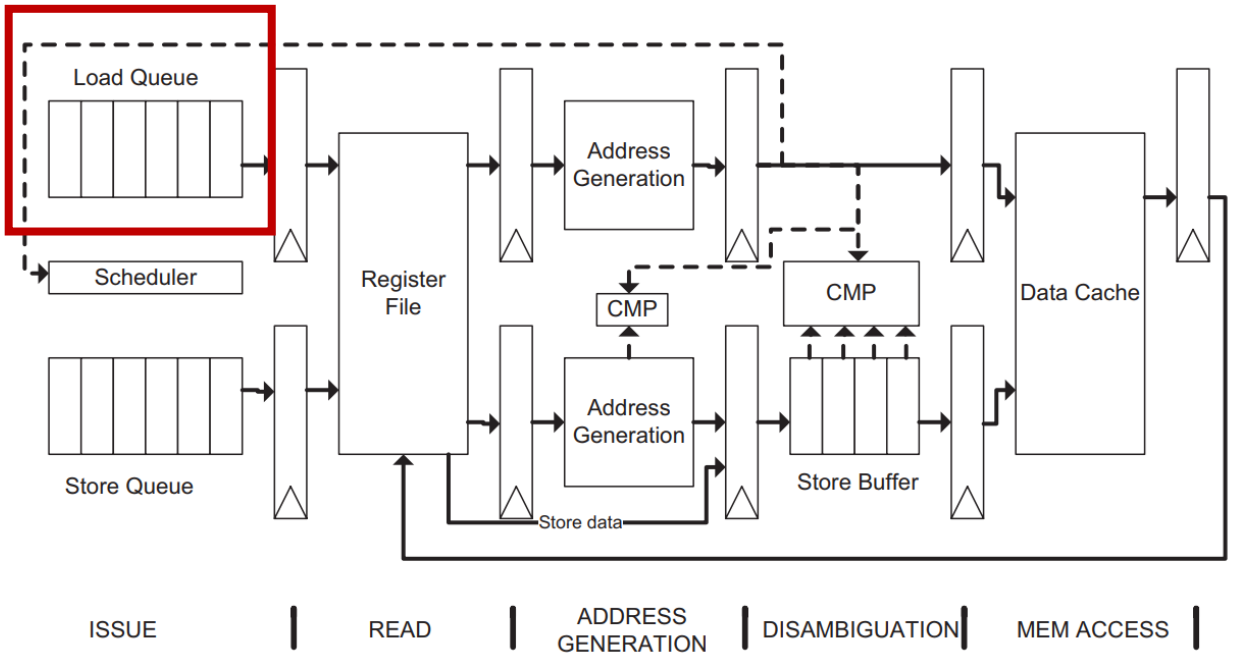
Load Ordering and Store Ordering on AMD K6

- two separate pipelines for load and store operations w/ some level of communication
- instructions flow in strict order inside each pipeline



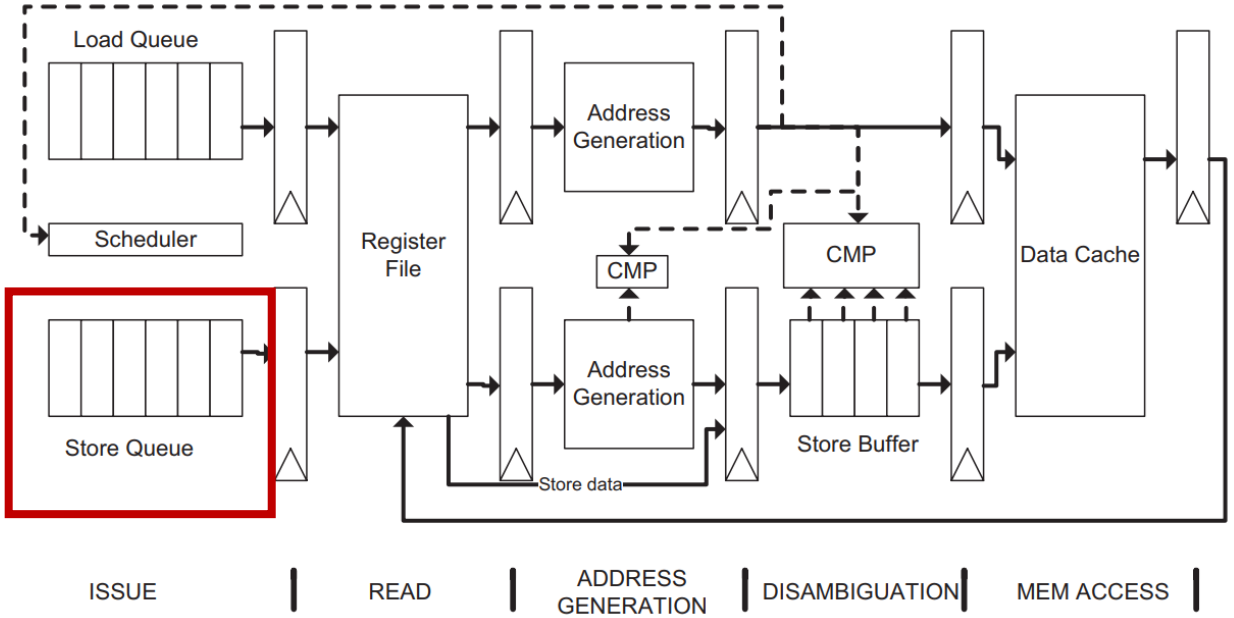
Load Ordering and Store Ordering on AMD K6

- Load queue: stores LD operations in order
  - LD operations are inserted in this queue after renaming and reside there until they become oldest on queue and their source operands are ready.



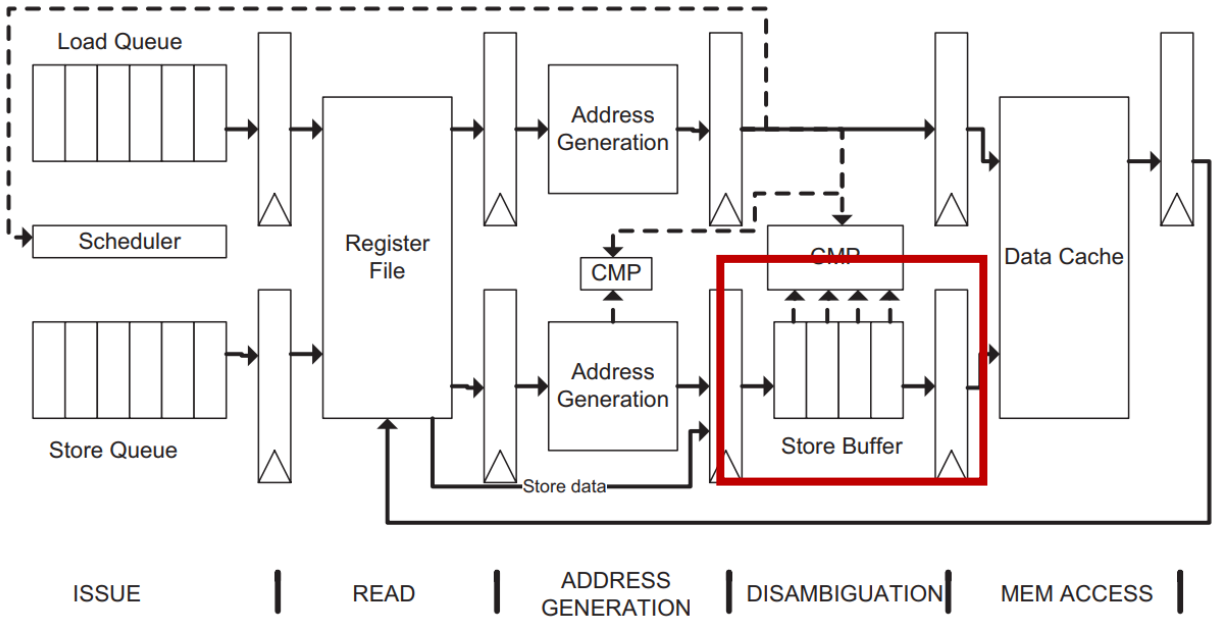
Load Ordering and Store Ordering on AMD K6

- Store queue: stores the ST operations in order
  - ST operations reside here since they have been renamed until they become oldest instruction on queue; source operands they need to compute the address are available



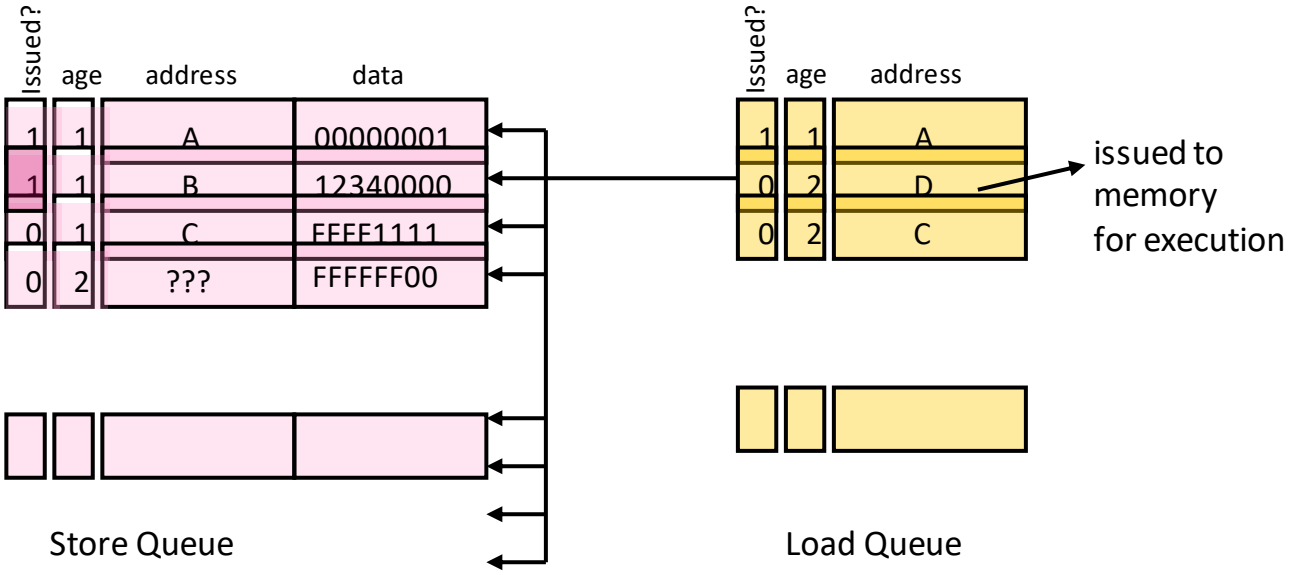
Load Ordering and Store Ordering on AMD K6

- Store buffer: keeps the ST operations in order until they become oldest in-flight instruction in the processor, and then they proceed to update the memory



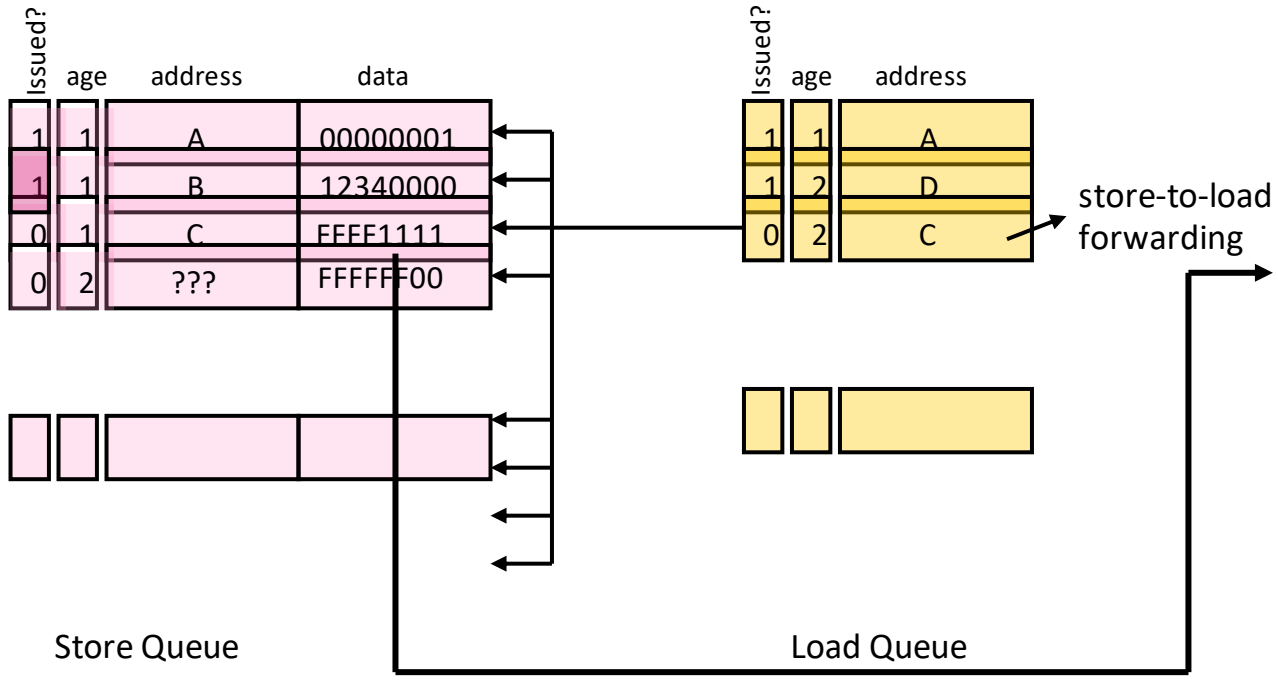
LSQ

- load will compare its memory address with addresses of stores in store buffer that are older than it
- each load checks against older stores
  - associative search → performance scalability issue



LSQ

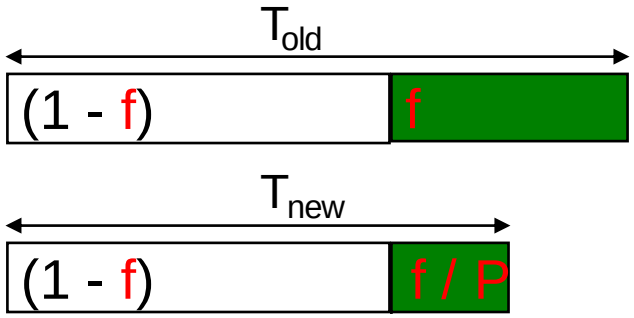
- each load checks against older stores
  - associative search → performance scalability issue



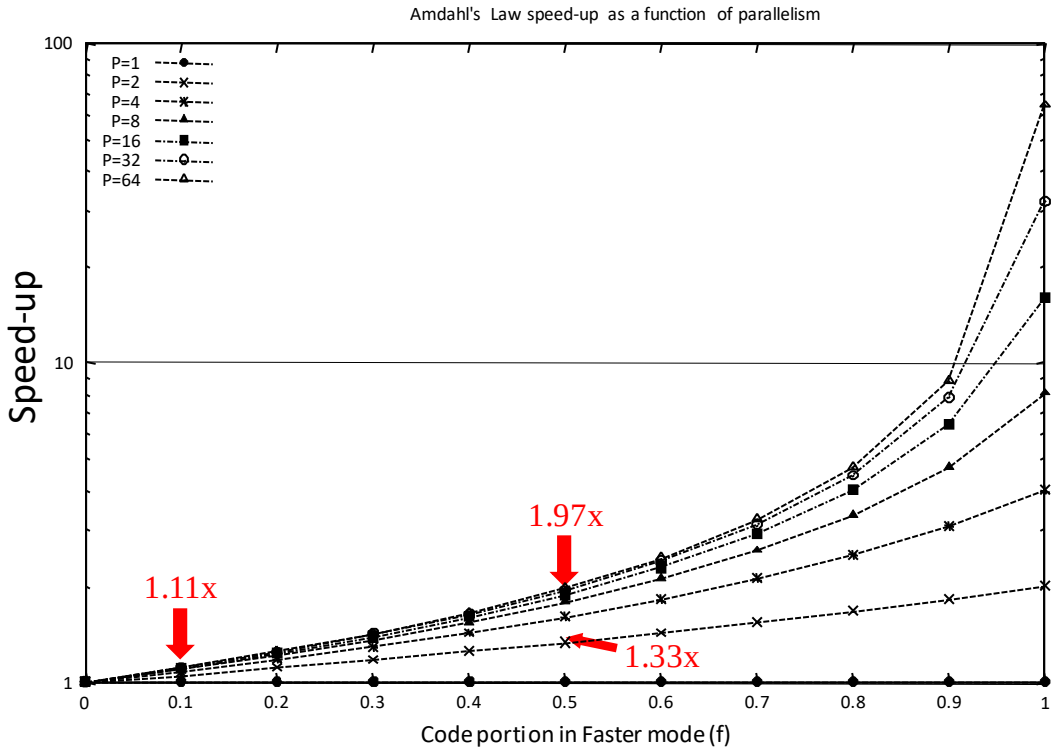
Amdahl's Law

- law of diminishing returns
  - make the common case faster
  - speedup = $Perf_{new} / Perf_{old} = T_{old} / T_{new} =$ $\dfrac{1}{(1-f)+\frac{f}{P}}$

- Example
  - Suppose floating point instructions occupy10% of a program, they are improved to run 2X faster, what's the speedup?

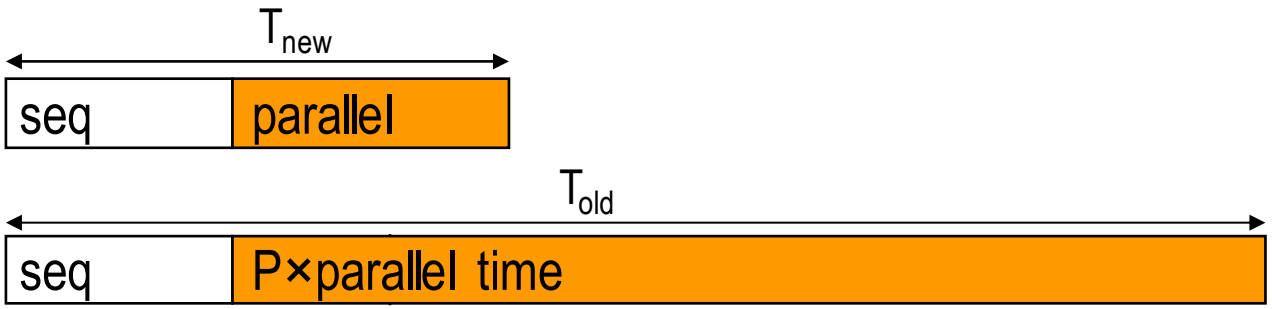


Parallelism vs. Speedup



Gustafson's Law

- Amdahl's Law killed massive parallel processing (MPP)
- Gustafson came to rescue
  - ✓ more workloads

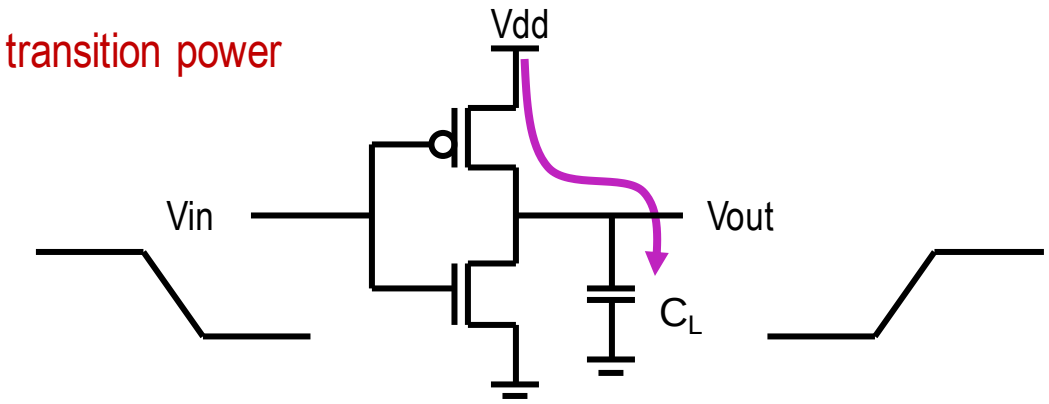


- ✓ assume: seq + parallel = 1 (Tnew)
- ✓ speedup = seq + p × (1 – seq) where p = parallel factor
- ✓ if seq diminishes w/ increased problem size, speedup → p

New Breed of Metrics

- performance/Watt
  - performance achievable at the same cooling capacity
- performance/Joule (energy)
  - achievable performance at the lifetime of the same energy source (i.e., battery = energy)
  - equivalent to reciprocal of energy-delay product (the product of E and D)

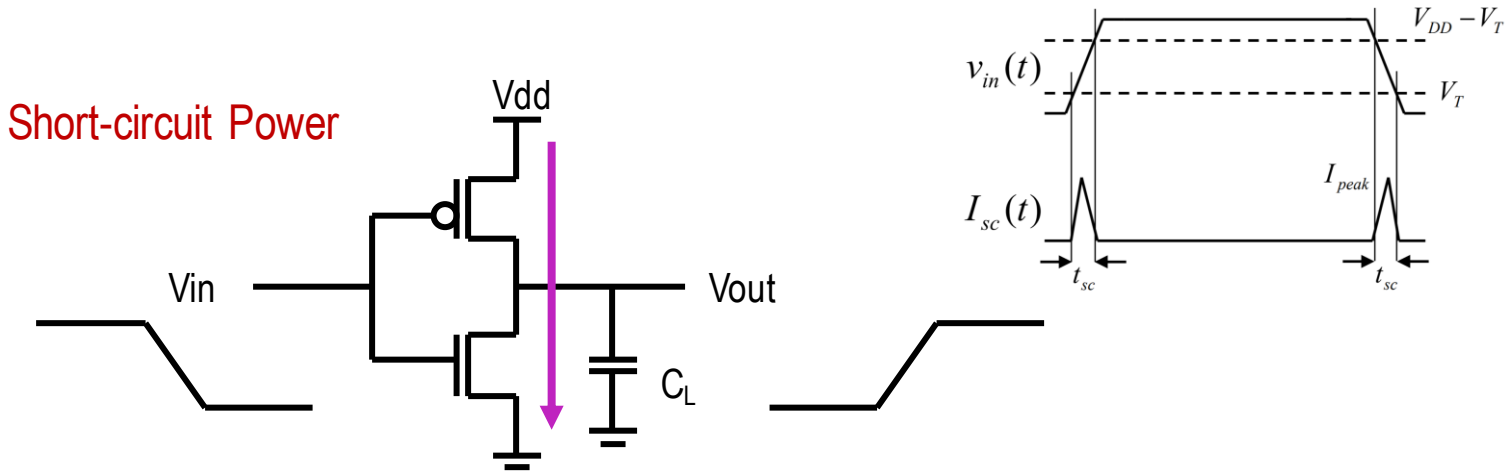
(Dynamic) Power Dissipation



$$\text{energy/transition} = C_L \times V_{DD}^2 \times P_{0/1 \rightarrow 1/0}$$

$$\text{power} = C_L \times V_{DD}^2 \times f$$

(Short Circuit) Power Dissipation



Energy/transition = $t_{sc} \times V_{DD} \times I_{peak} \times P_{0/1 \rightarrow 1/0}$

Power = $t_{sc} \times V_{DD} \times I_{peak} \times f$

CMOS Energy & Power Equations

$$E = C_L V_{DD}^2 P_{0\to1} + t_{sc} V_{DD} I_{peak} P_{0\to1} + V_{DD} I_{leakage}$$

$$f_{0\to1} = P_{0\to1} * f_{clock}$$

$$P = C_L V_{DD}^2 f_{0\to1} + t_{sc} V_{DD} I_{peak} f_{0\to1} + V_{DD} I_{leakage}$$

dynamic power (≈ 40 - 70% today and decreasing relatively)

short-circuit power (≈ 10 % today and decreasing absolutely)

leakage power (≈ 20 – 50 % today and increasing)

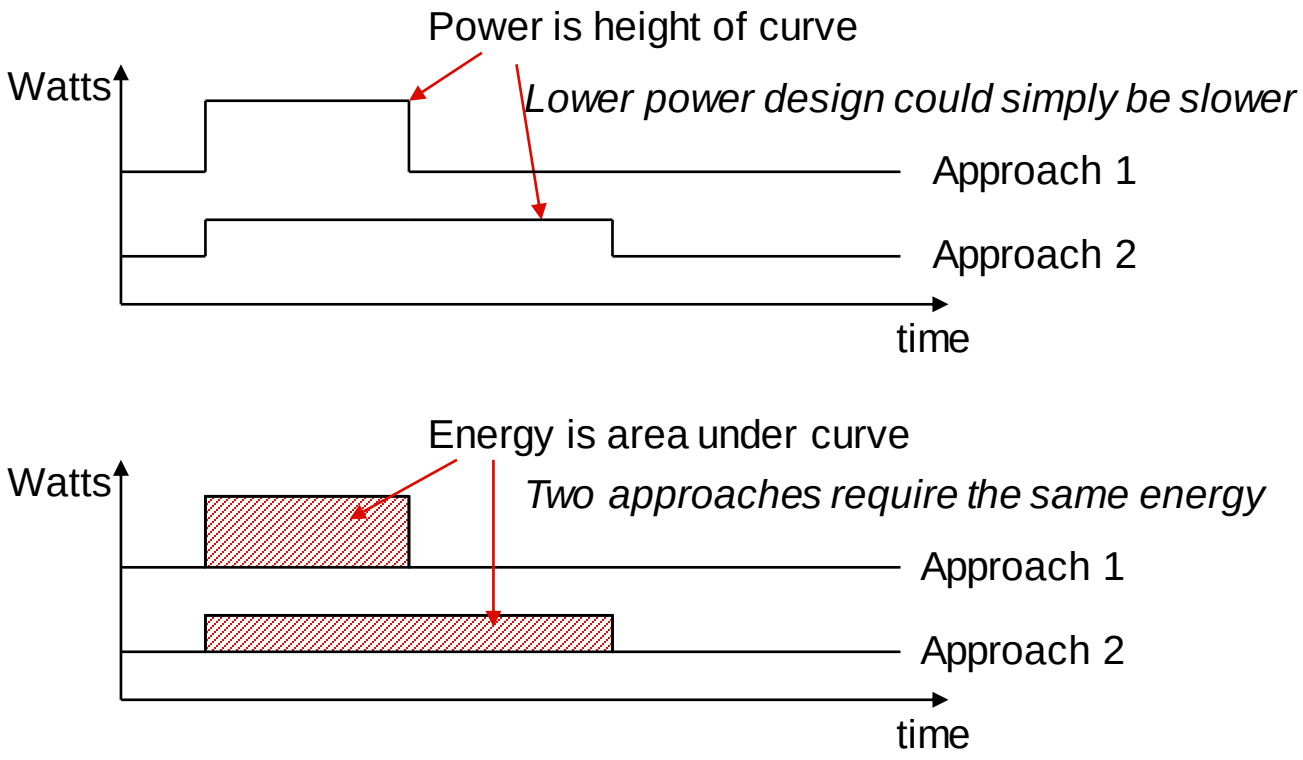
Power and Energy Metrics

- power in Watts
  - **Watt = Energy / Time**
  - poses constraints:
    - i.e., processor can only work fast enough to max out the power delivery or cooling solution
  - Peak power
    - determines power ground wiring designs
    - sets packaging/cooling limits
    - impacts signal noise margin and reliability analysis

Power and Energy Metrics

- Energy in Joules
  - ultimate metric, i.e., the true “cost” of performing a fixed task
  - energy = power × delay
    - Joules = Watts × seconds
    - Lower energy means less power to perform a computation at the same frequency
- Example:
  - if processor A consumes 1.2× power of processor B, but finishes the task in 30% less time, which processor is more energy efficient and by how much?

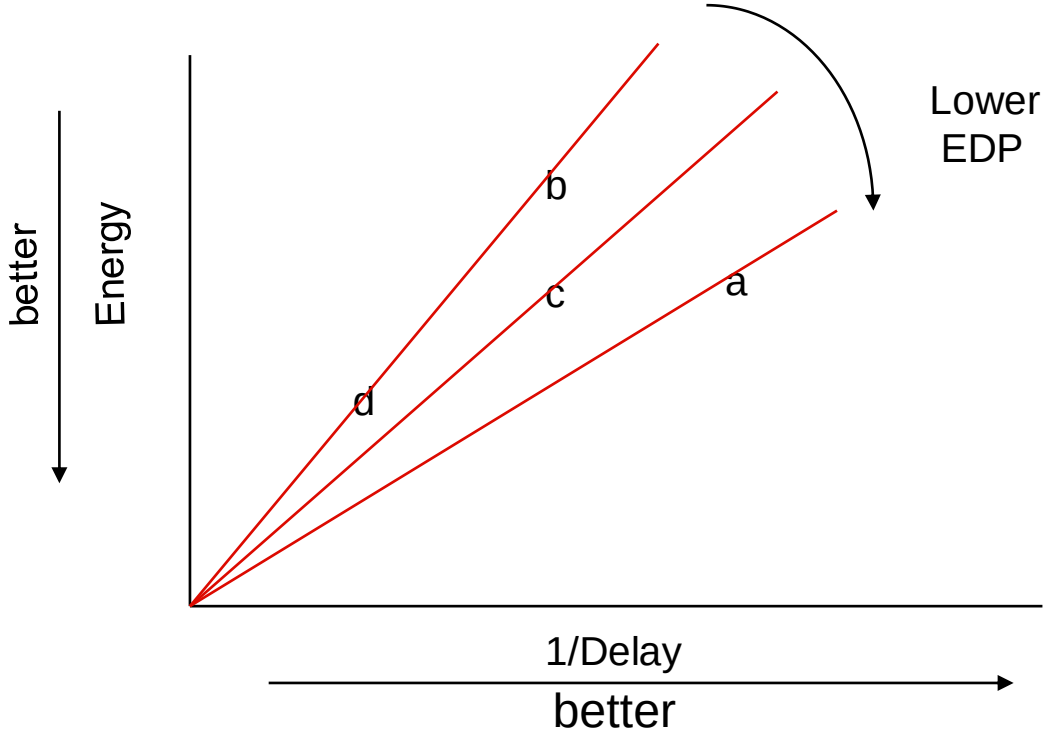
Power versus Energy



More Energy Metrics

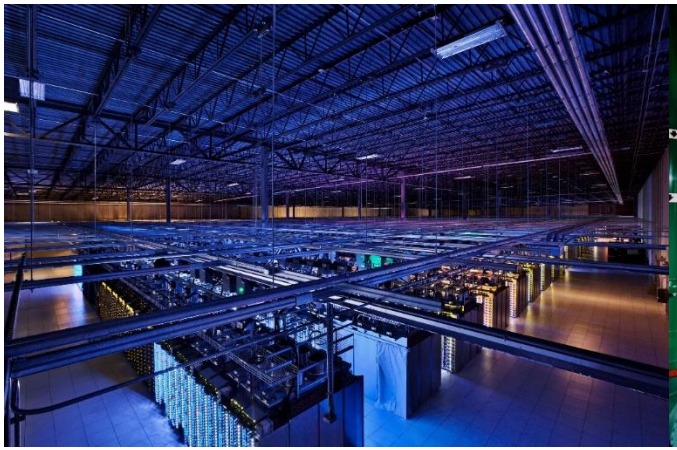
EDP = Energy Delay Product = E x D

EDDP = Energy Delay^2 Product = ED^2 (more emphasis on performance)

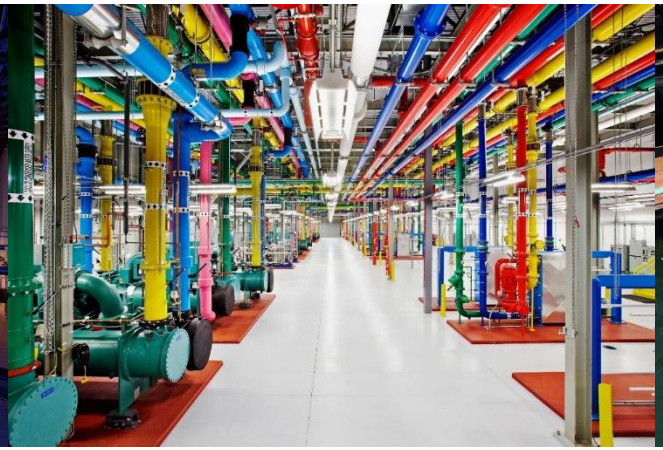


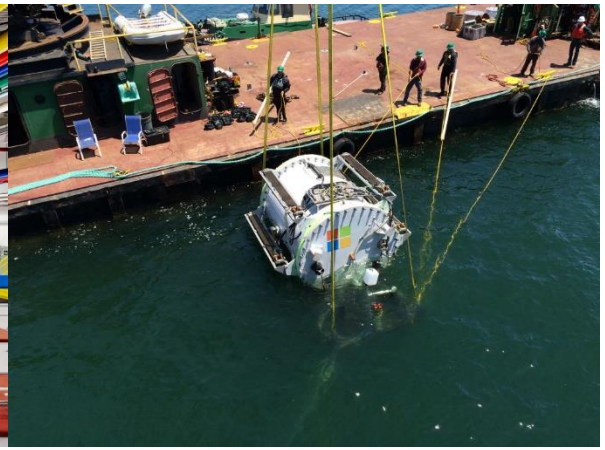
Research on Power/Energy

- What happens when a CPU heatsink is removed
  - Check this YouTube link



Google Data Center Cooling System





Microsoft Experimental Datacenter