EECS 583 – Class 17
Automatic Parallelization Via Decoupled Software Pipelining

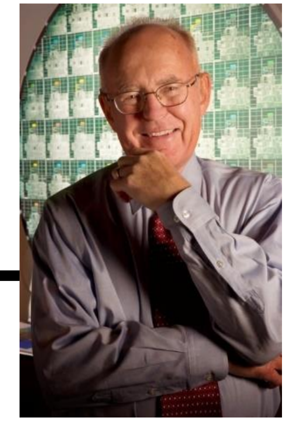
University of Michigan

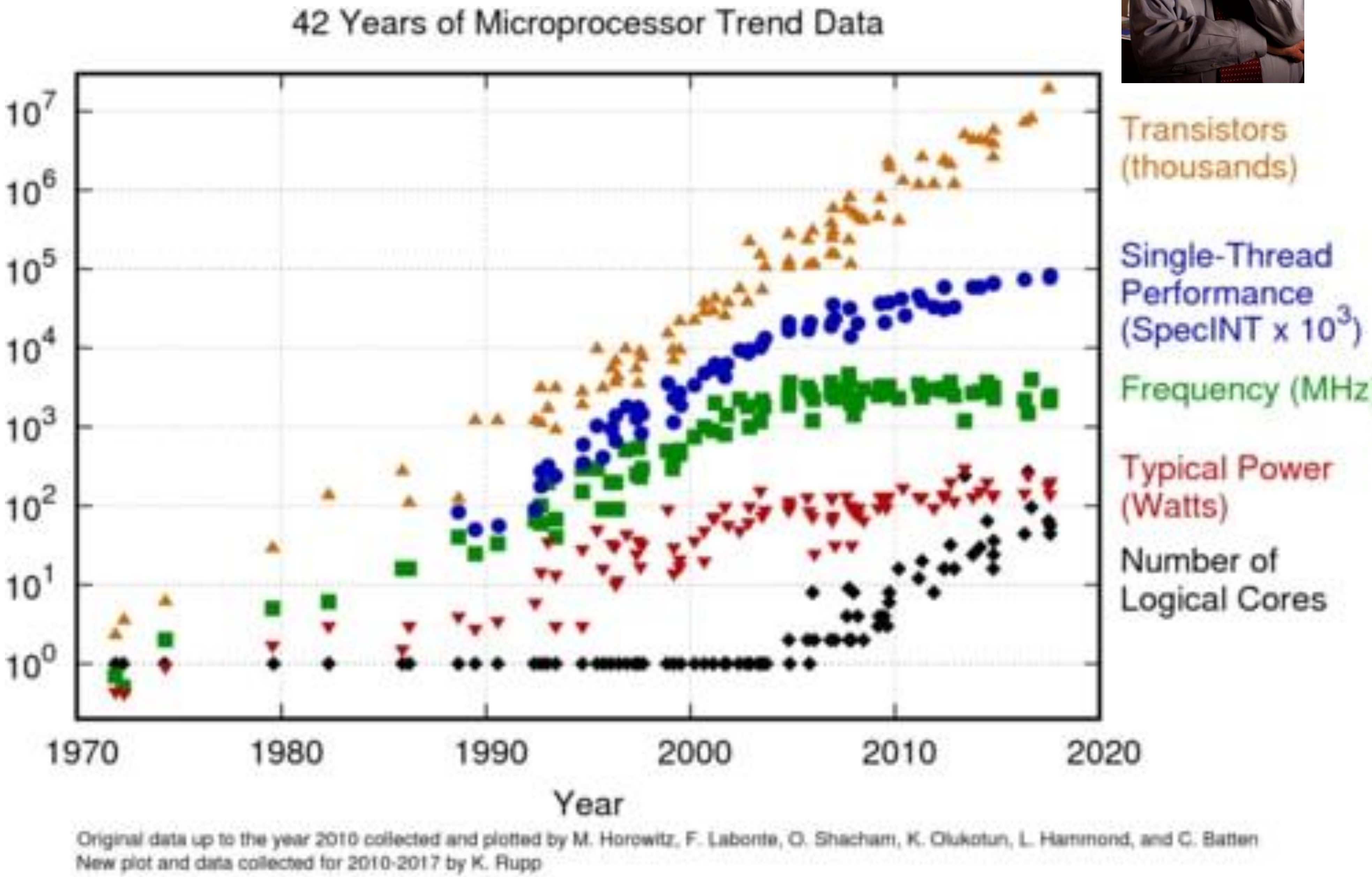
November 10, 2021

Announcements + Reading Material

- Research paper presentations
  - 3 more today + my last presentation today
  - Fill out quizzes (feedback forms) on canvas
- Reading material
  - "Revisiting the Sequential Programming Model for Multi-Core," M. J. Bridges, N. Vachharajani, Y. Zhang, T. Jablin, and D. I. August, Proc 40th IEEE/ACM International Symposium on Microarchitecture, December 2007.
  - "Automatic Thread Extraction with Decoupled Software Pipelining," G. Ottoni, R. Rangan, A. Stoler, and D. I. August, *Proceedings of the 38th IEEE/ACM International Symposium on Microarchitecture*, Nov. 2005

Moore's Law

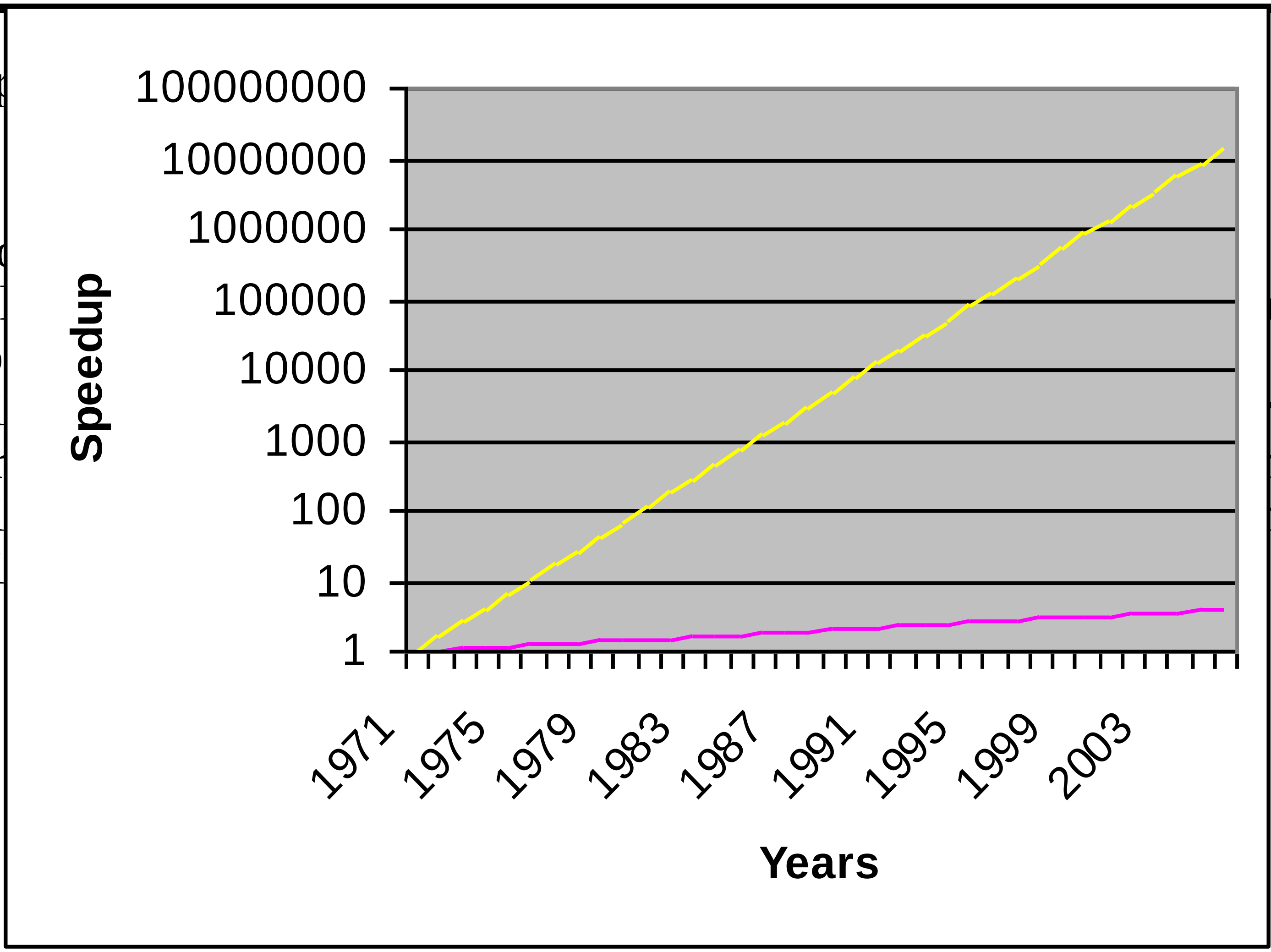



42 Years of Microprocessor Trend Data
Transistors (thousands)
Single-Thread Performance (SpecINT x 10^3)
Frequency (MHz)
Typical Power (Watts)
Number of Logical Cores
10^7
10^6
10^5
10^4
10^3
10^2
10^1
10^0
1970
1980
1990
2000
2010
2020
Year
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

Compilers are the Answer? - Proebsting's Law

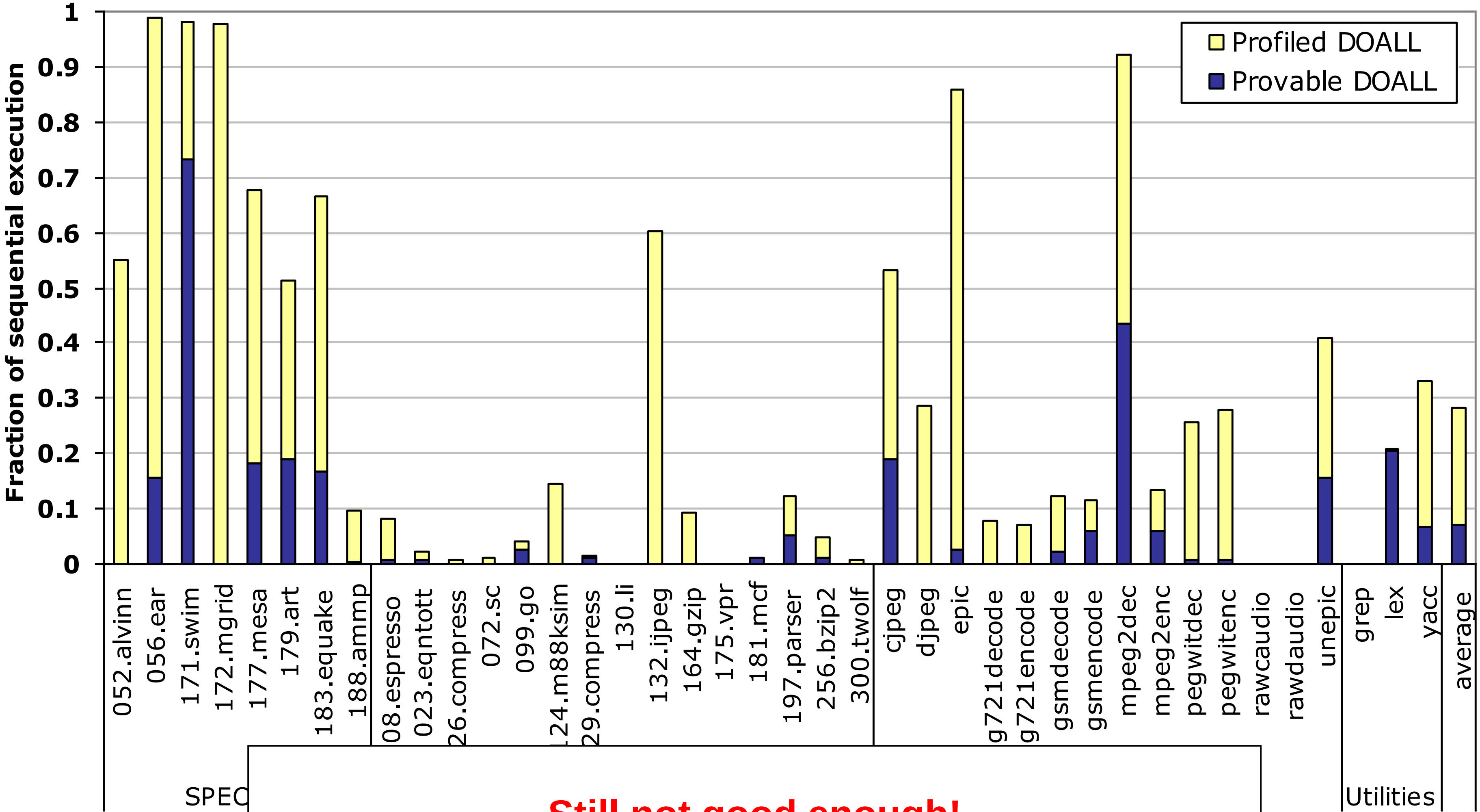
- "Comp

- Run yo … optimizing compil … bled. The ratio o … ompiler optimi … atio is about 4X for … piler optimi … piler optimi



Conclusion – Compilers not about performance!

DOALL Coverage – Provable and Profiled



Still not good enough!

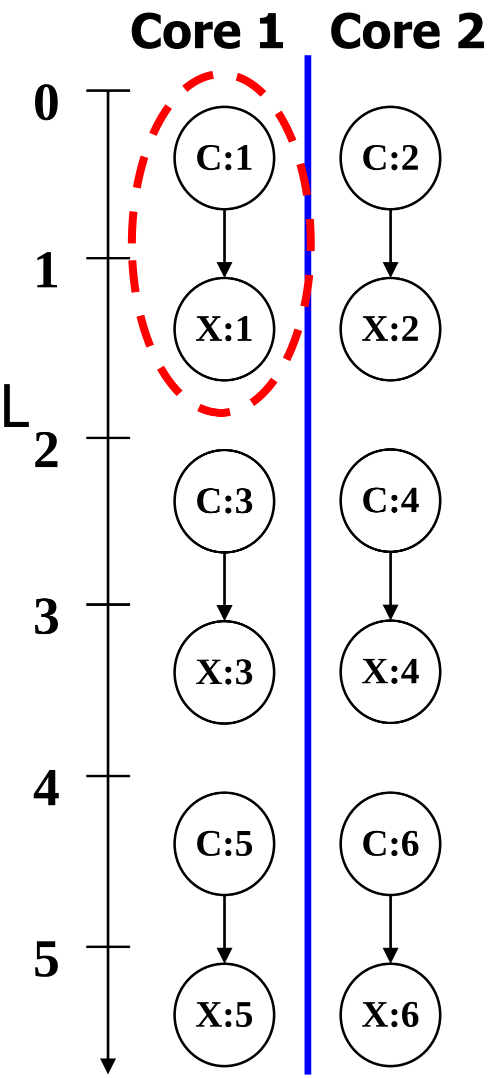
What About Non-Scientific Codes???

Scientific Codes (FORTRAN-like)

```
for(i=1; i<=N; i++)  // C
  a[i] = a[i] + 1;   // X
```

Independent Multithreading (IMT)

Example: DOALL parallelization

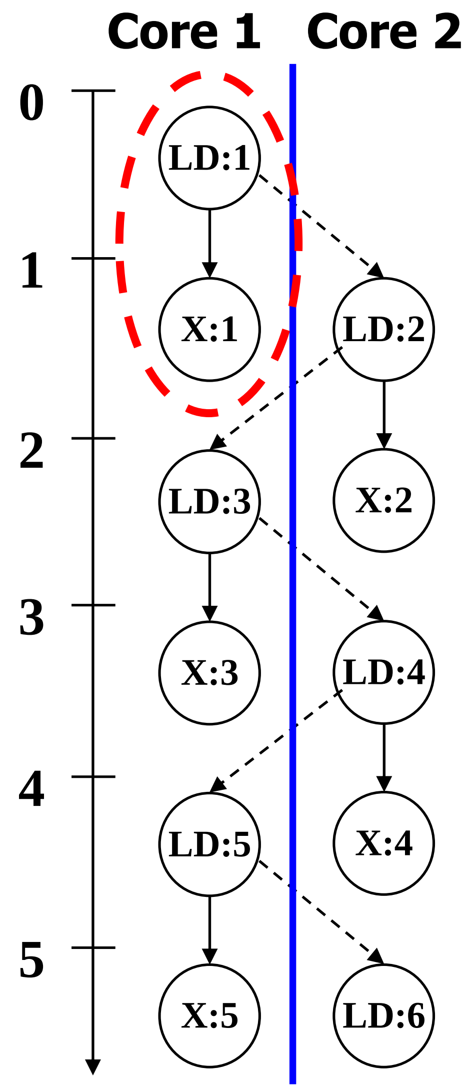


General-purpose Codes (legacy C/C++)

```
while(ptr = ptr->next)       // LD
  ptr->val = ptr->val + 1; // X
```

Cyclic Multithreading (CMT)

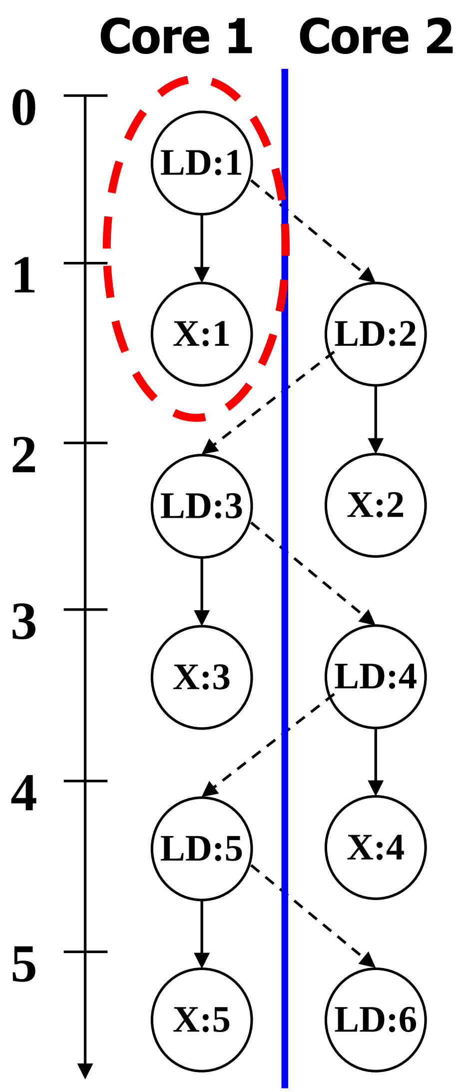
Example: DOACROSS [Cytron, ICPP 86]



Alternative Parallelization Approaches

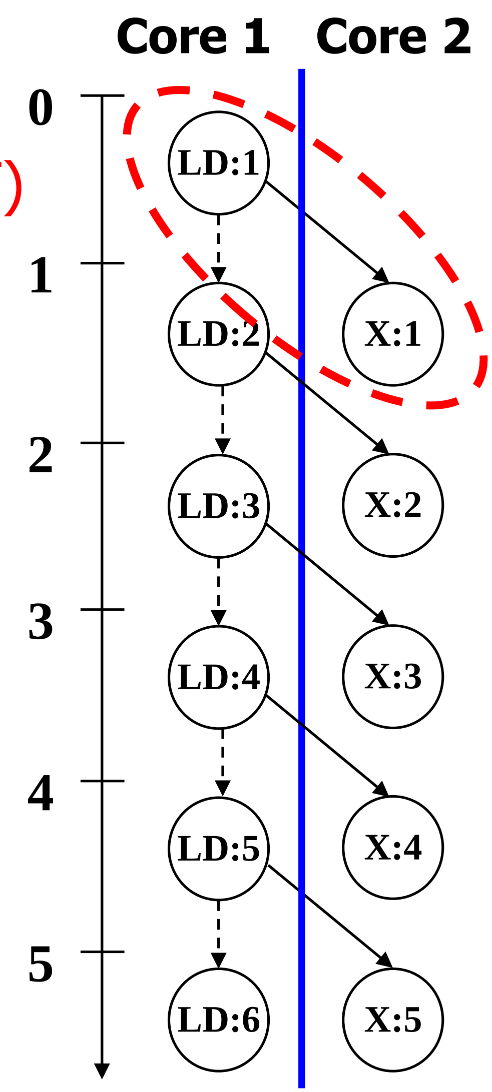
```
while(ptr = ptr->next)      // LD
  ptr->val = ptr->val + 1; // X
```

Cyclic Multithreading (CMT)

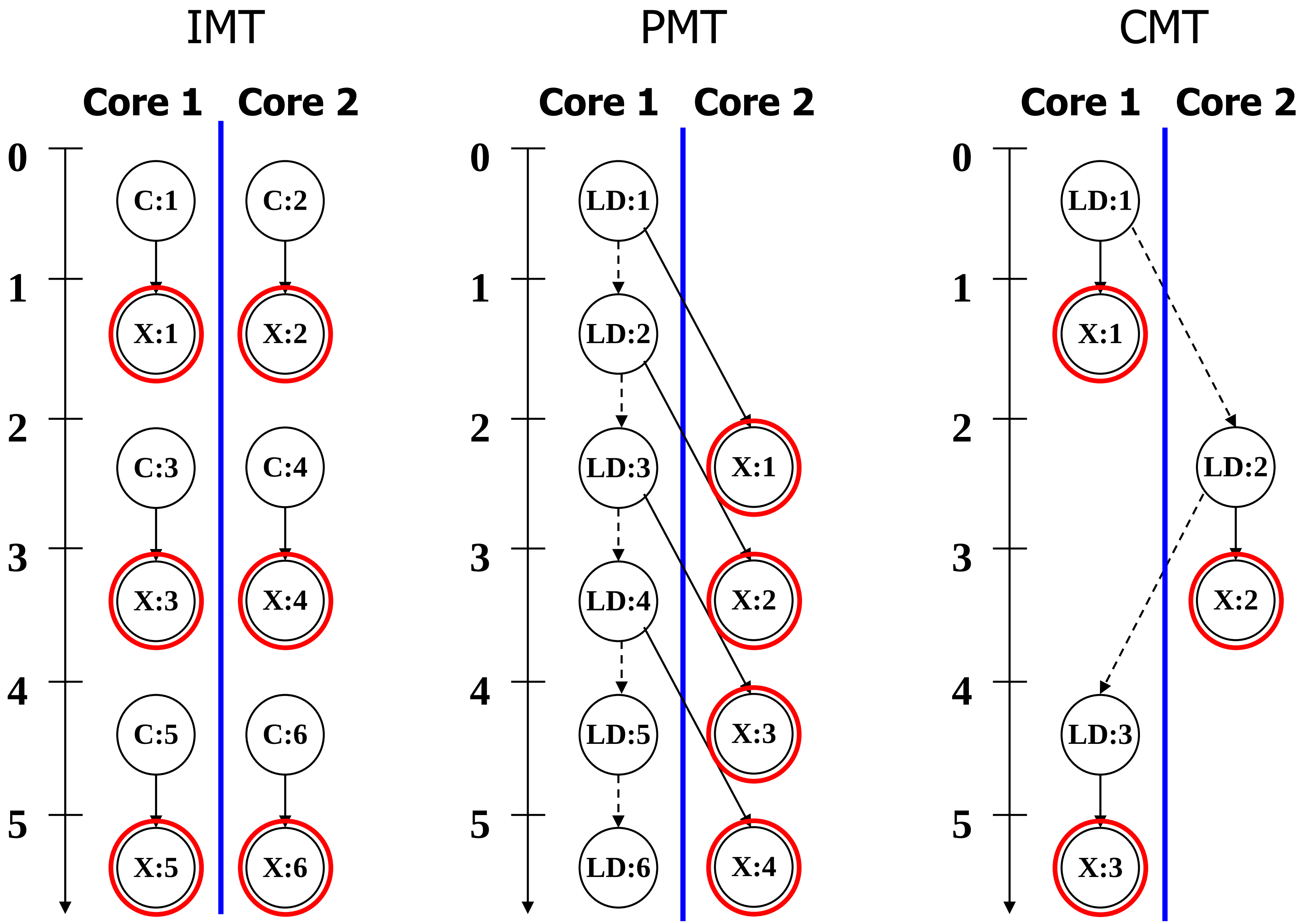


Pipelined Multithreading (PMT)

Example: DSWP **[PACT 2004]**



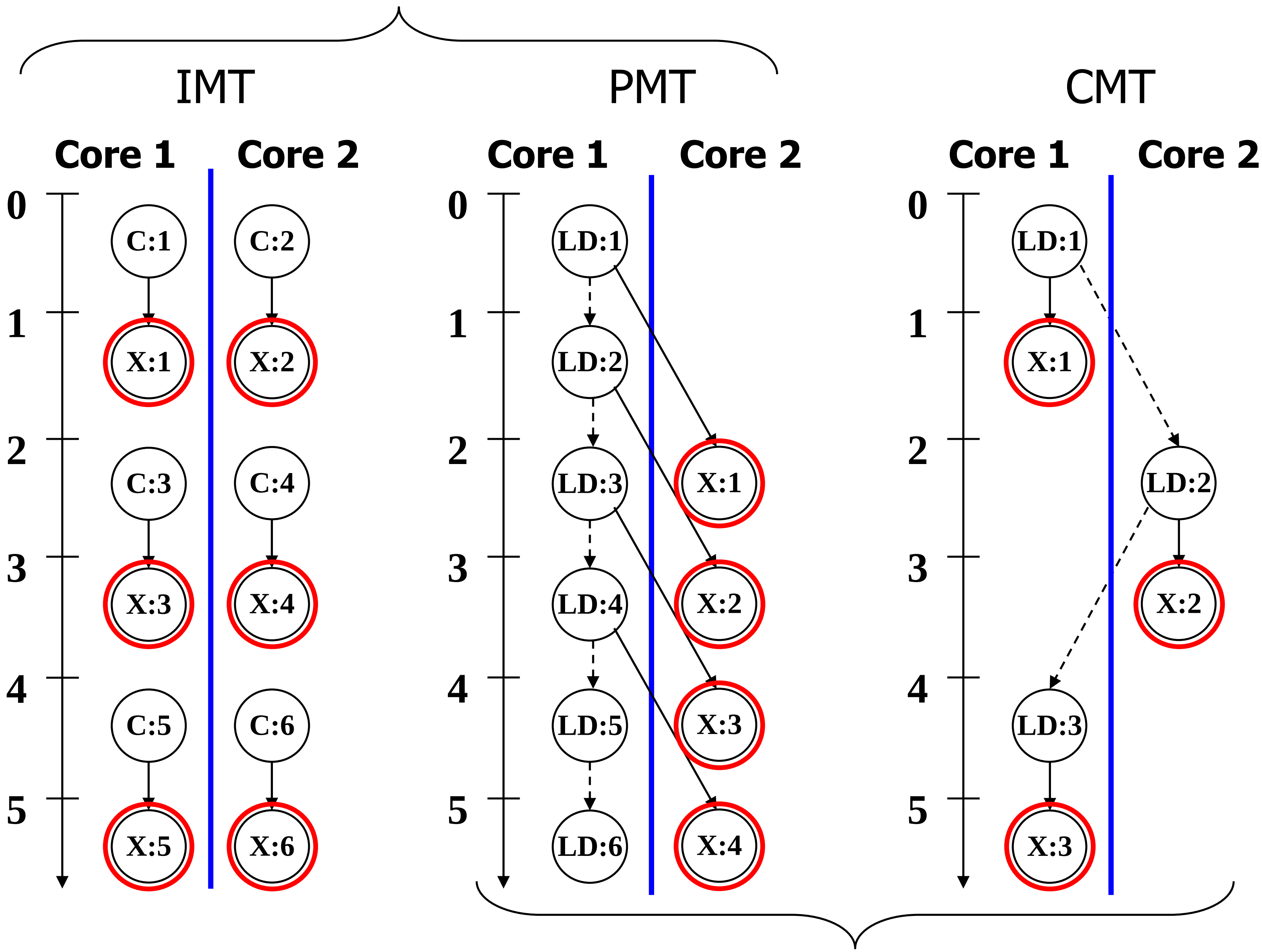
Comparison: IMT, PMT, CMT



	IMT	PMT	CMT
lat(comm) = 1:	1 iter/cycle	1 iter/cycle	1 iter/cycle
lat(comm) = 2:	1 iter/cycle	1 iter/cycle	0.5 iter/cycle

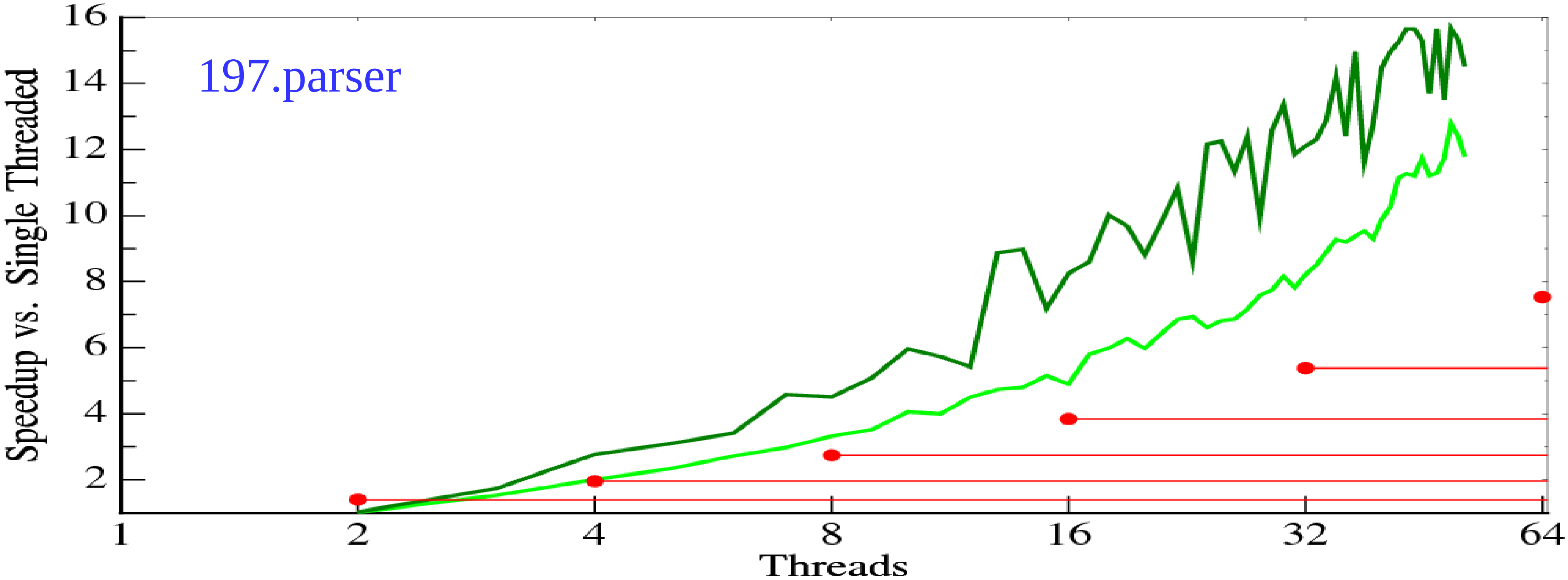
Comparison: IMT, PMT, CMT

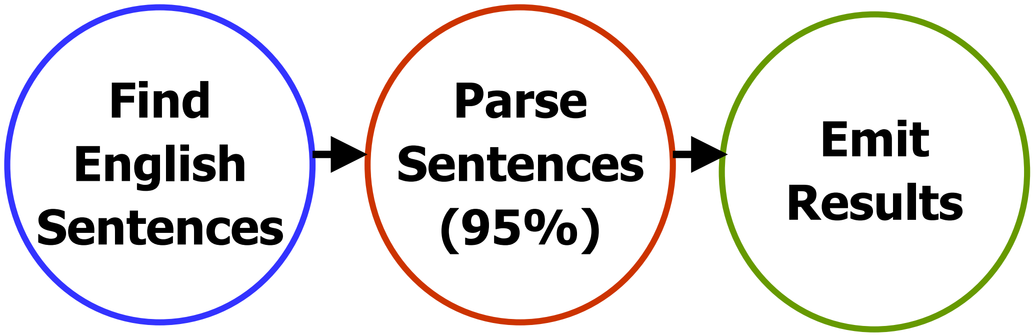
Thread-local Recurrences ➔ Fast Execution



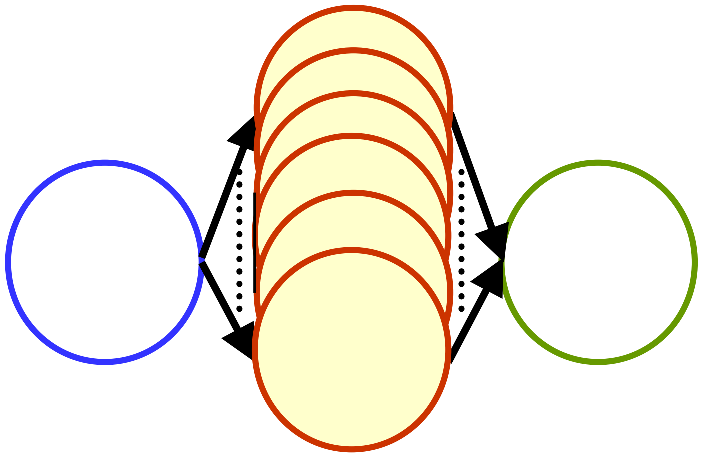
Cross-thread Dependences ➔ Wide Applicability

Our Objective: Automatic Extraction of Pipeline Parallelism using DSWP





Decoupled Software Pipelining



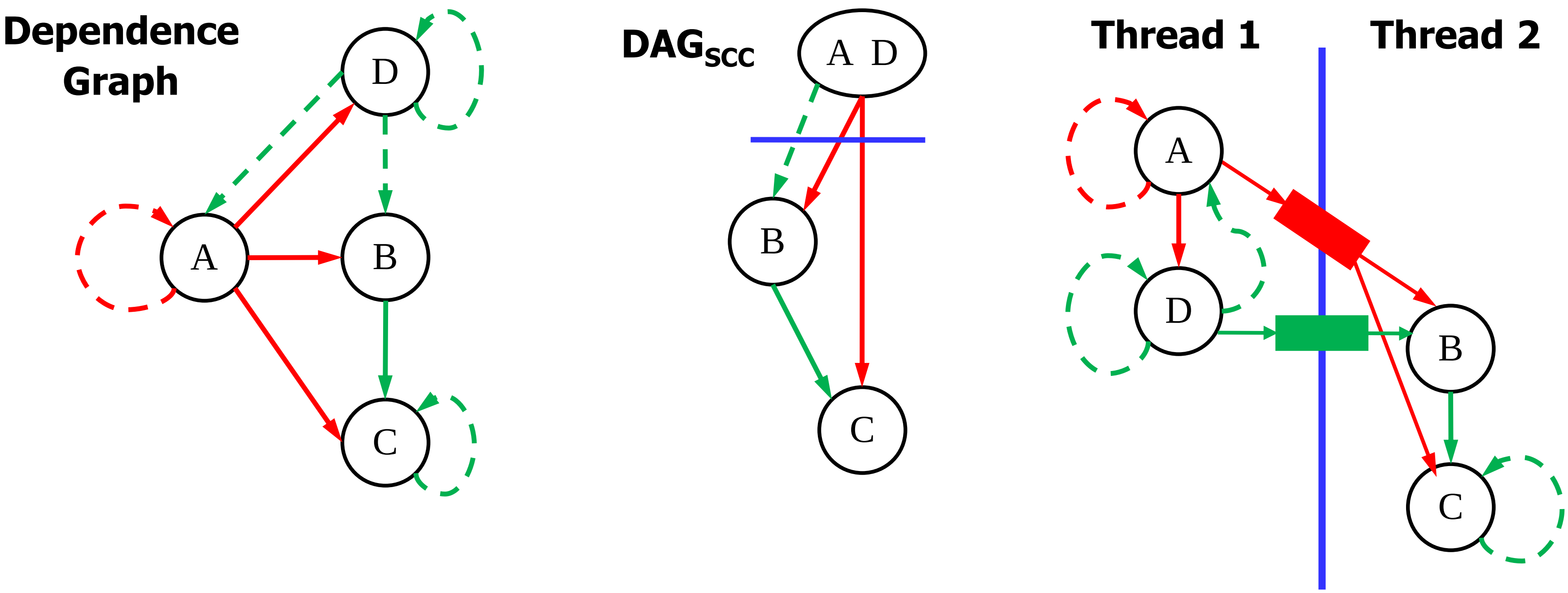
PS-DSWP (Spec DOALL Middle Stage)

Decoupled Software Pipelining

Decoupled Software Pipelining (DSWP)

[MICRO 2005]

```
A: while(node)
B:   ncost = doit(node);
C:   cost += ncost;
D:   node = node->next;
```



register

control

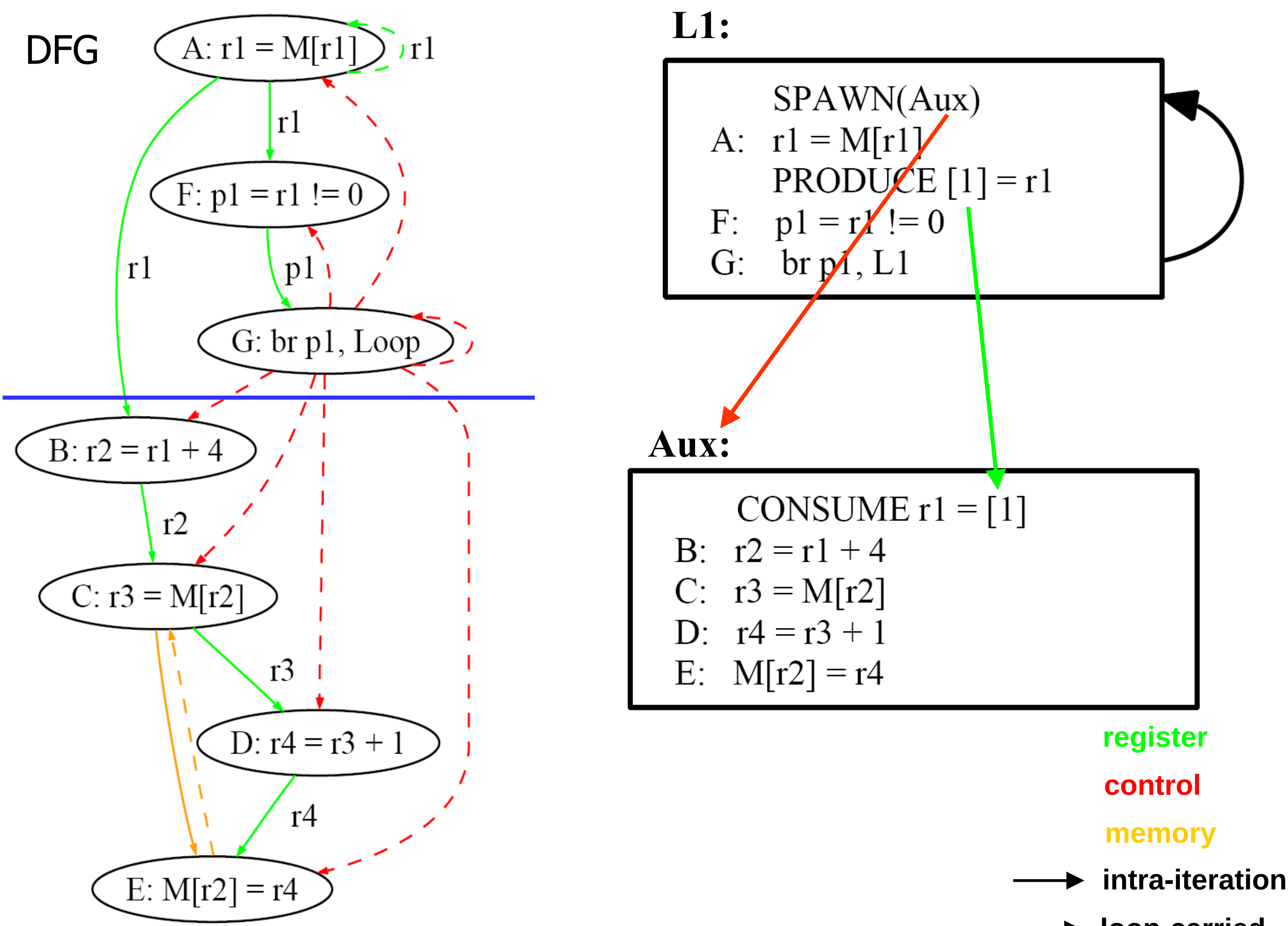
intra-iteration

loop-carried

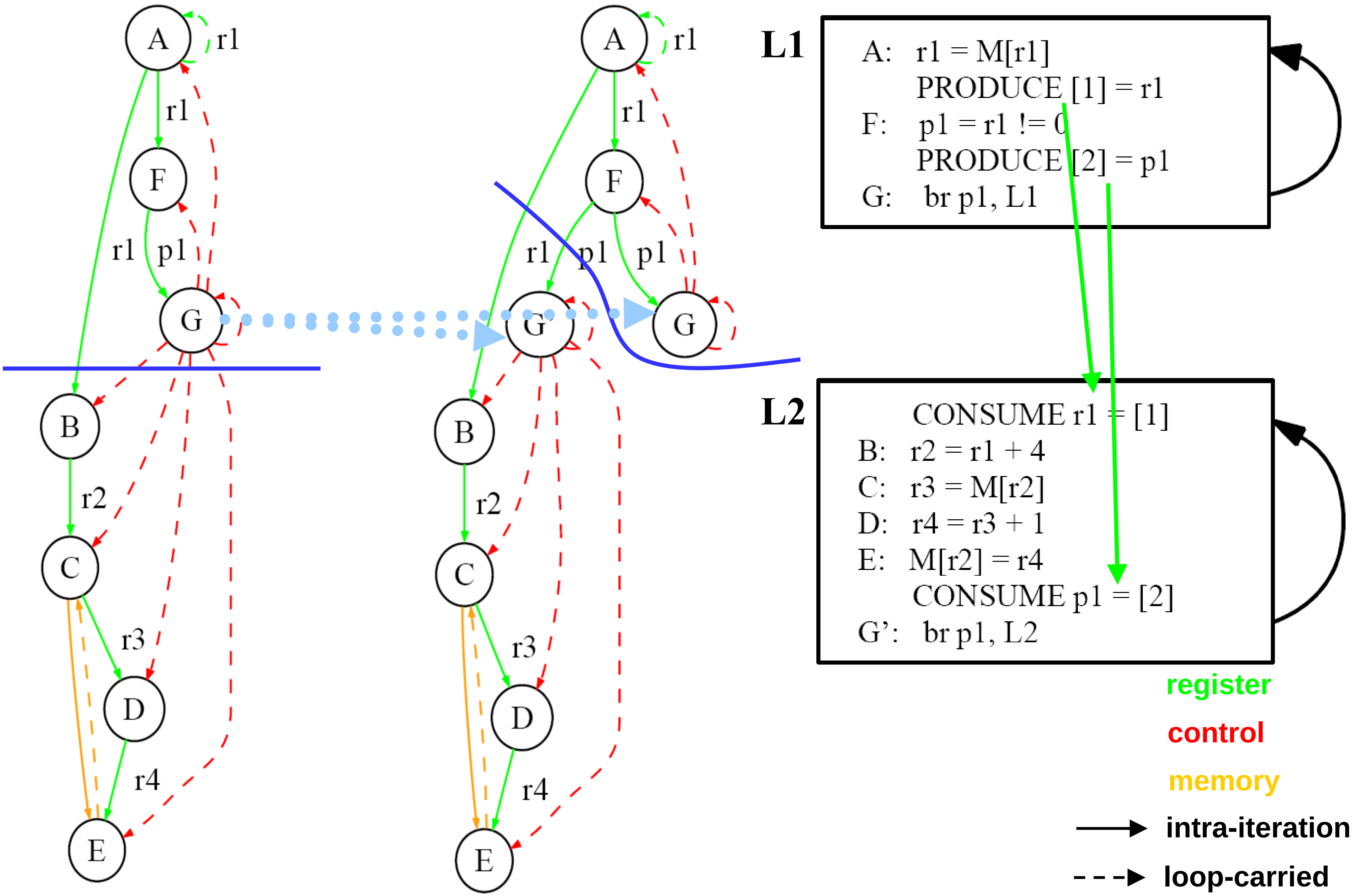
communication queue

Inter-thread communication latency is a one-time cost

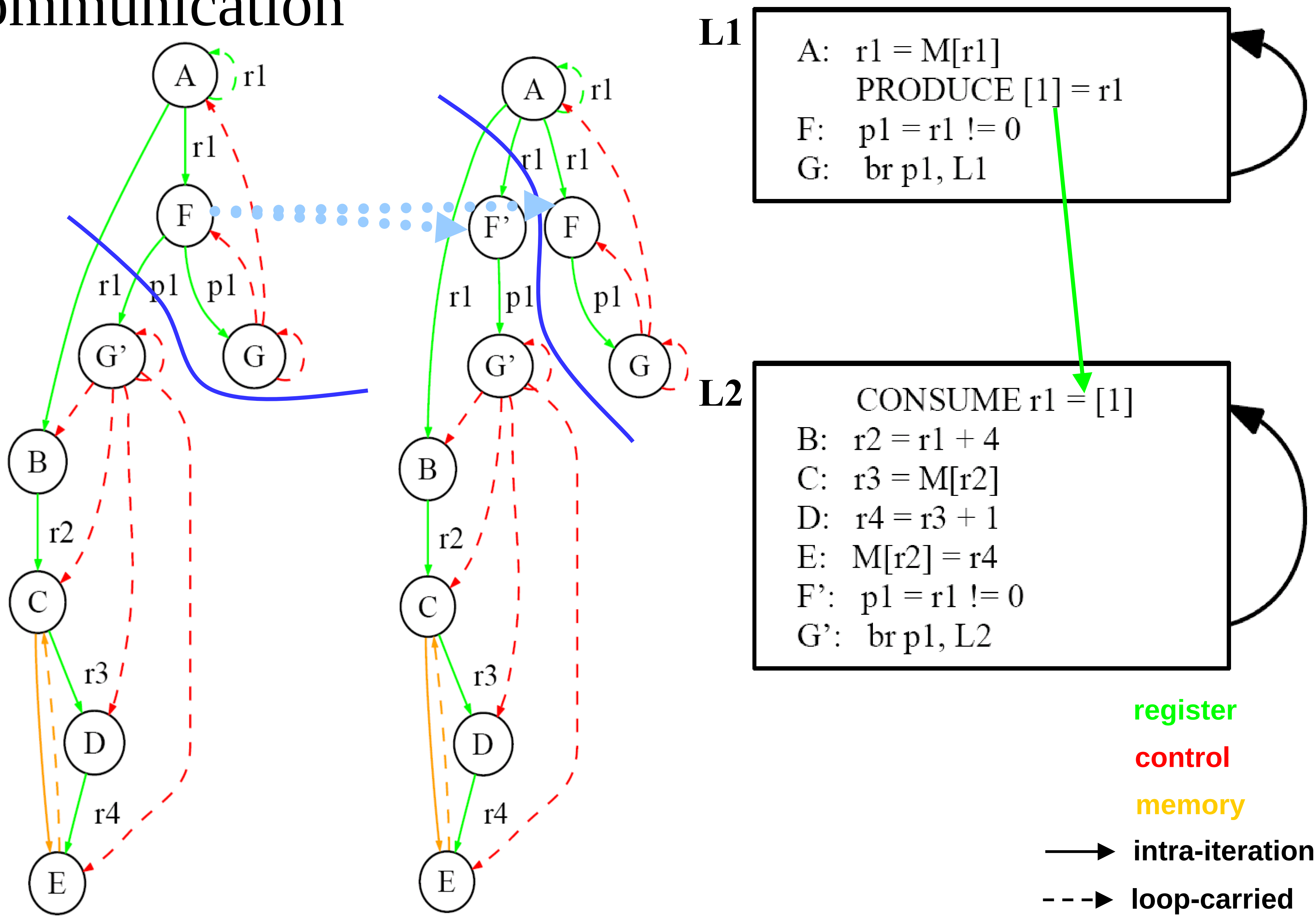
Implementing DSWP



Optimization: Node Splitting To Eliminate Cross Thread Control

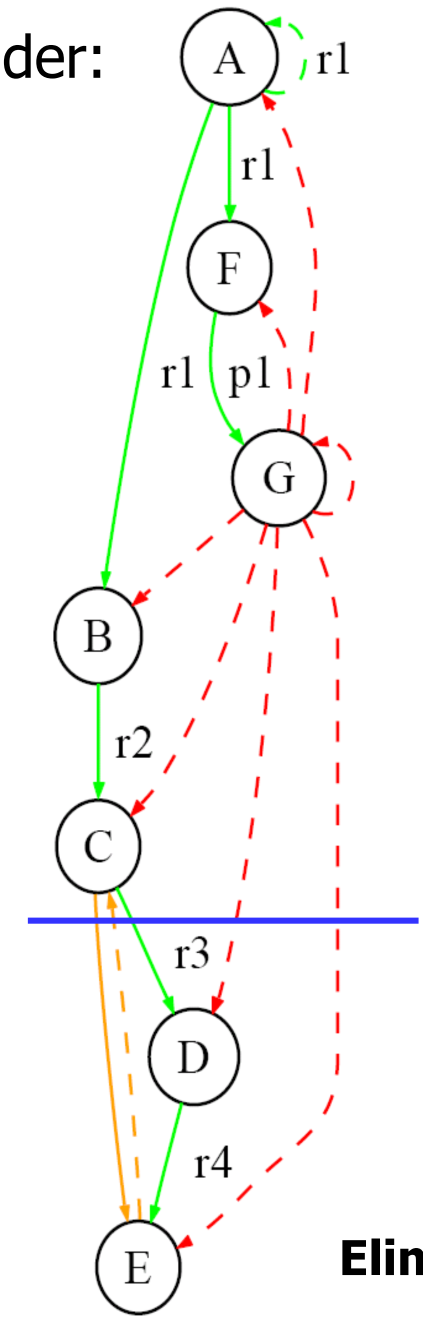


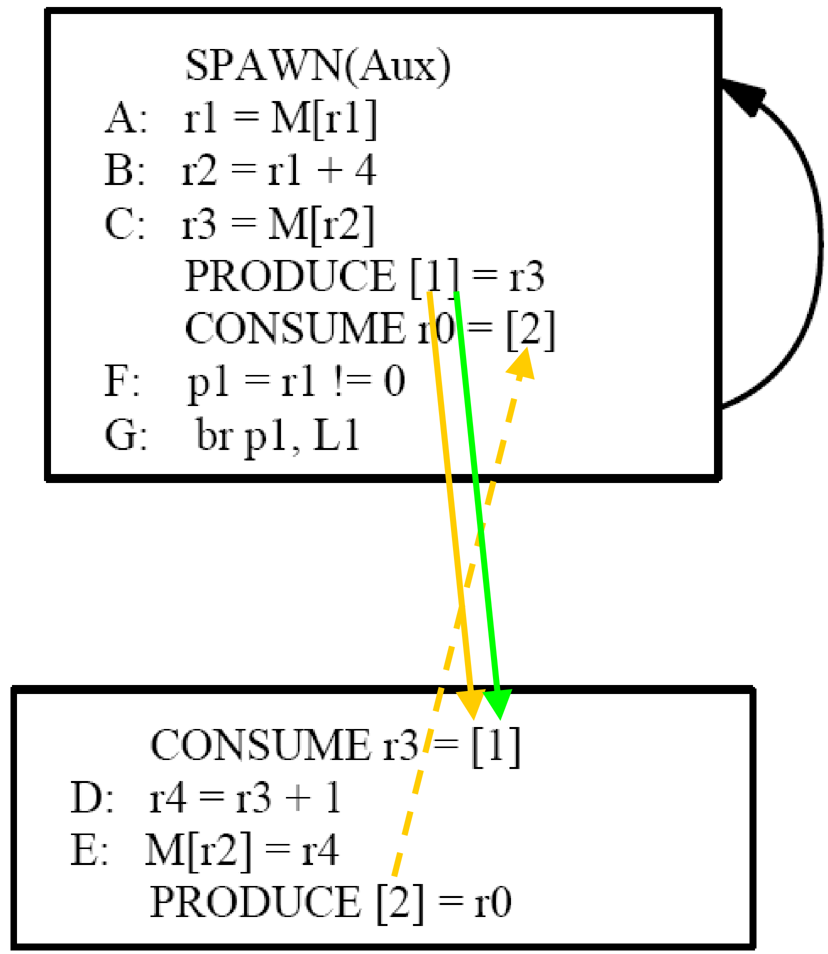
Optimization: Node Splitting To Reduce Communication



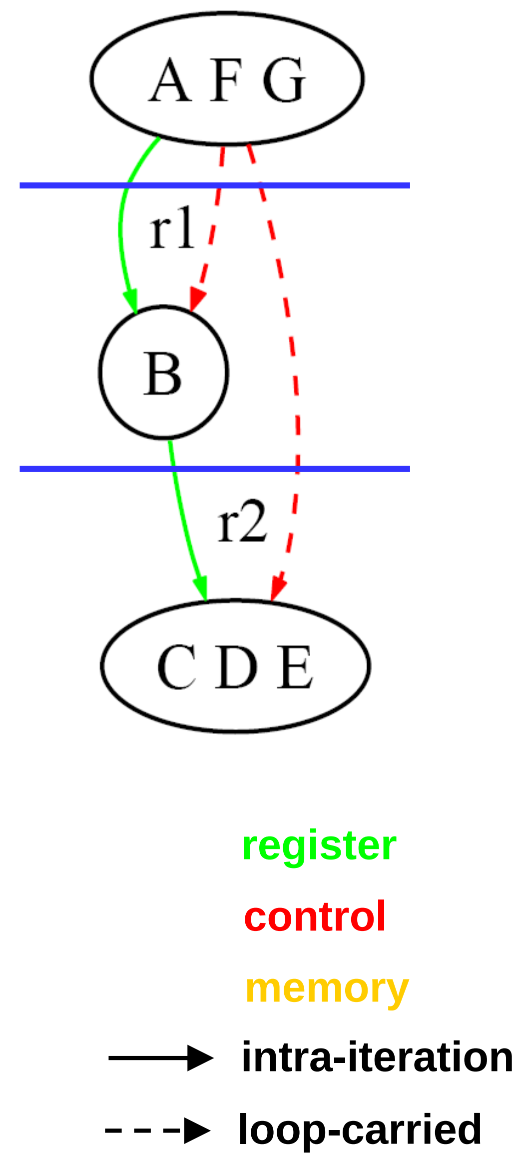
Constraint: Strongly Connected Components

Consider:





Solution: DAG_{SCC}



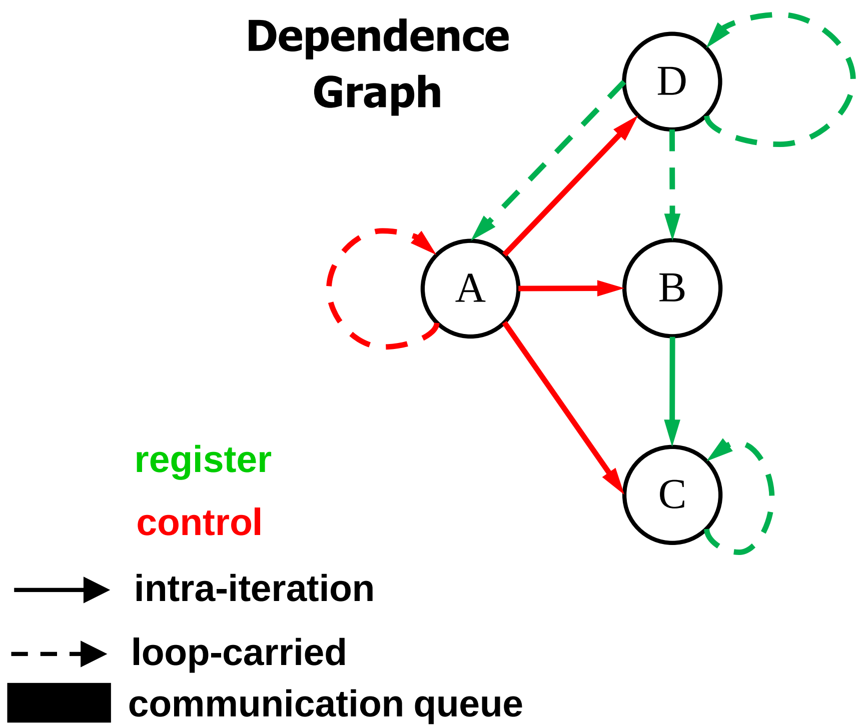
Eliminates pipelined/decoupled property

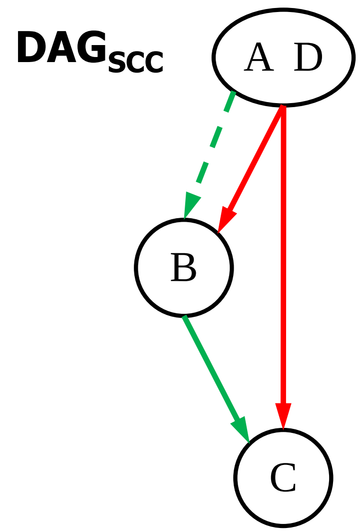
2 Extensions to the Basic Transformation

- Speculation
  - Break statistically unlikely dependences
  - Form better-balanced pipelines
- Parallel Stages
  - Execute multiple copies of certain “large” stages
  - Stages that contain inner loops perfect candidates

Why Speculation?

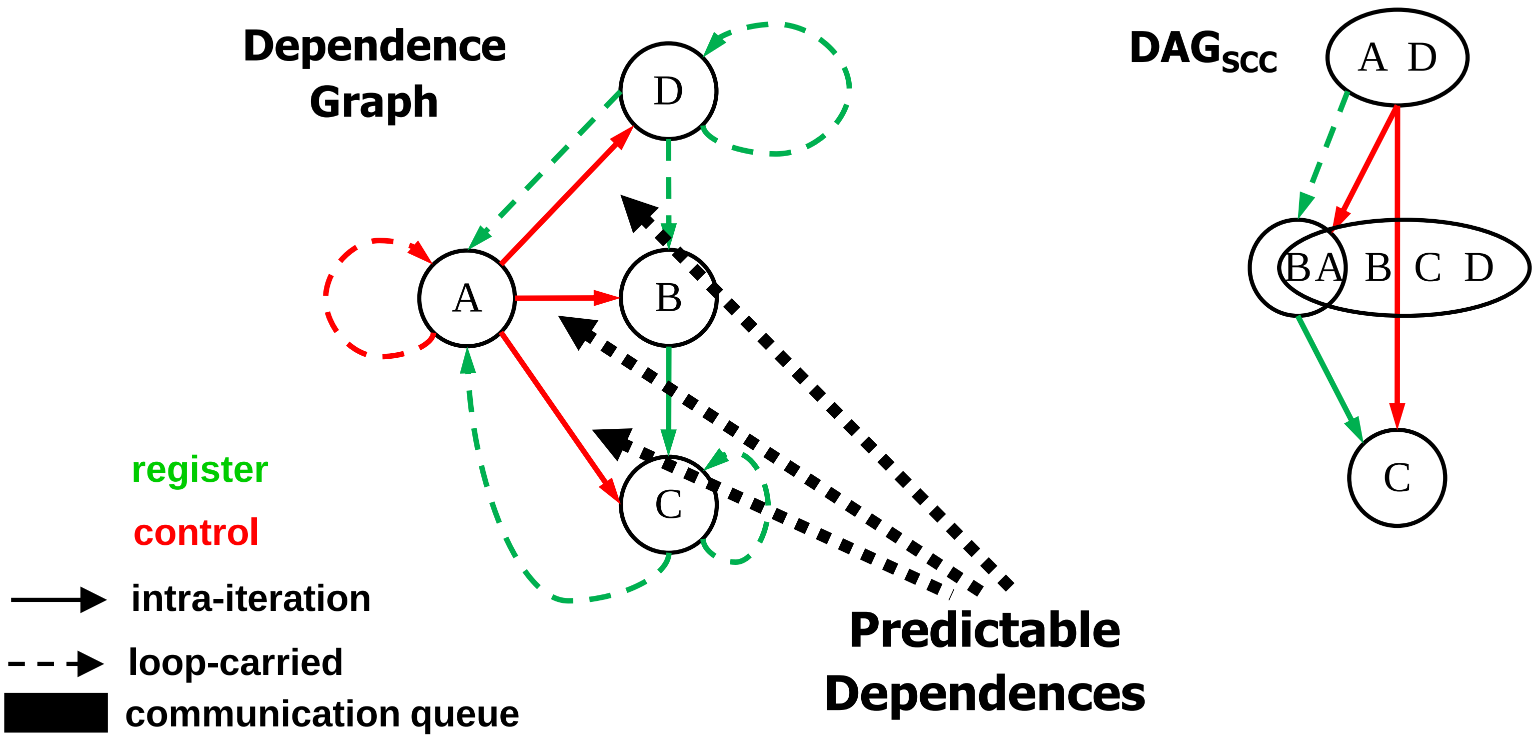
```
A:  while(node)
B:     ncost = doit(node);
C:     cost += ncost;
D:     node = node->next;
```





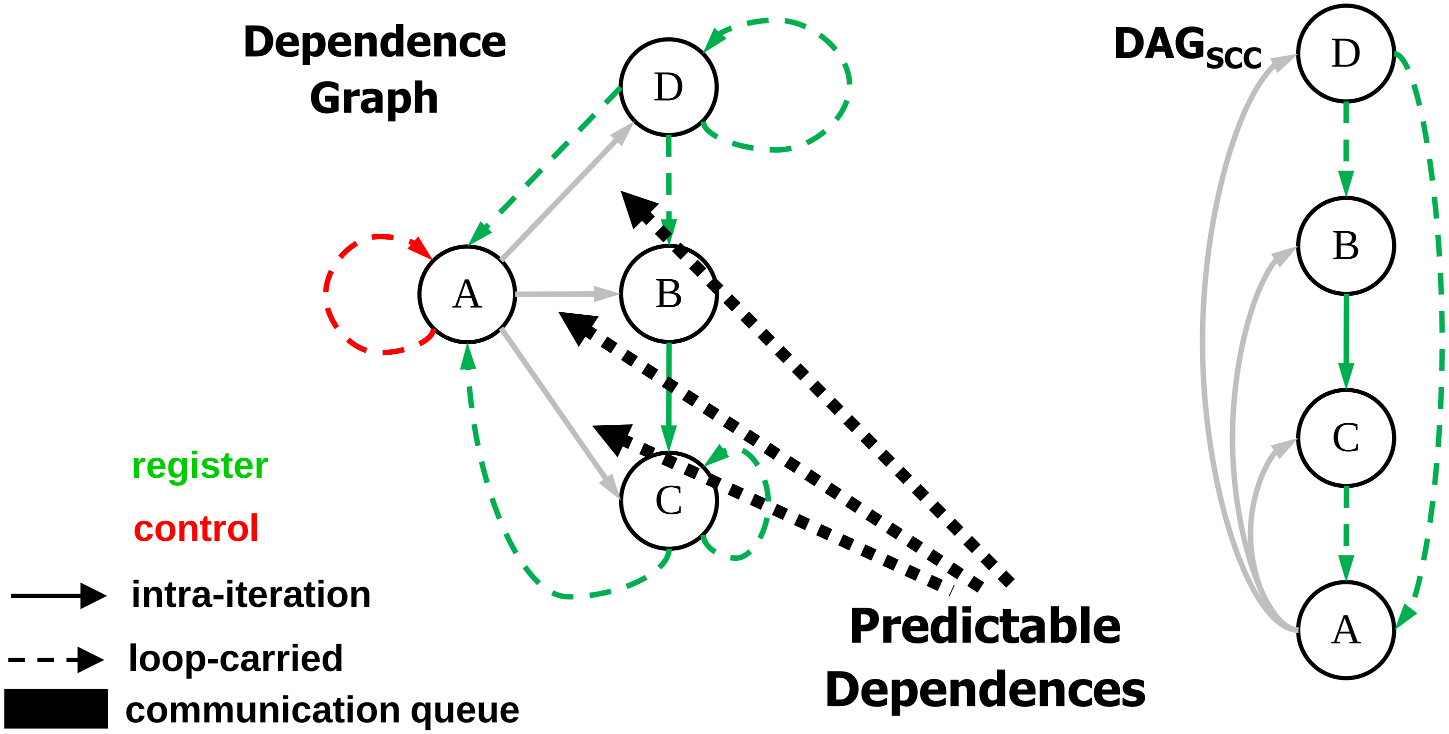
Why Speculation?

```
A: while(cost < T && node)
B:    ncost = doit(node);
C:    cost += ncost;
D:    node = node->next;
```

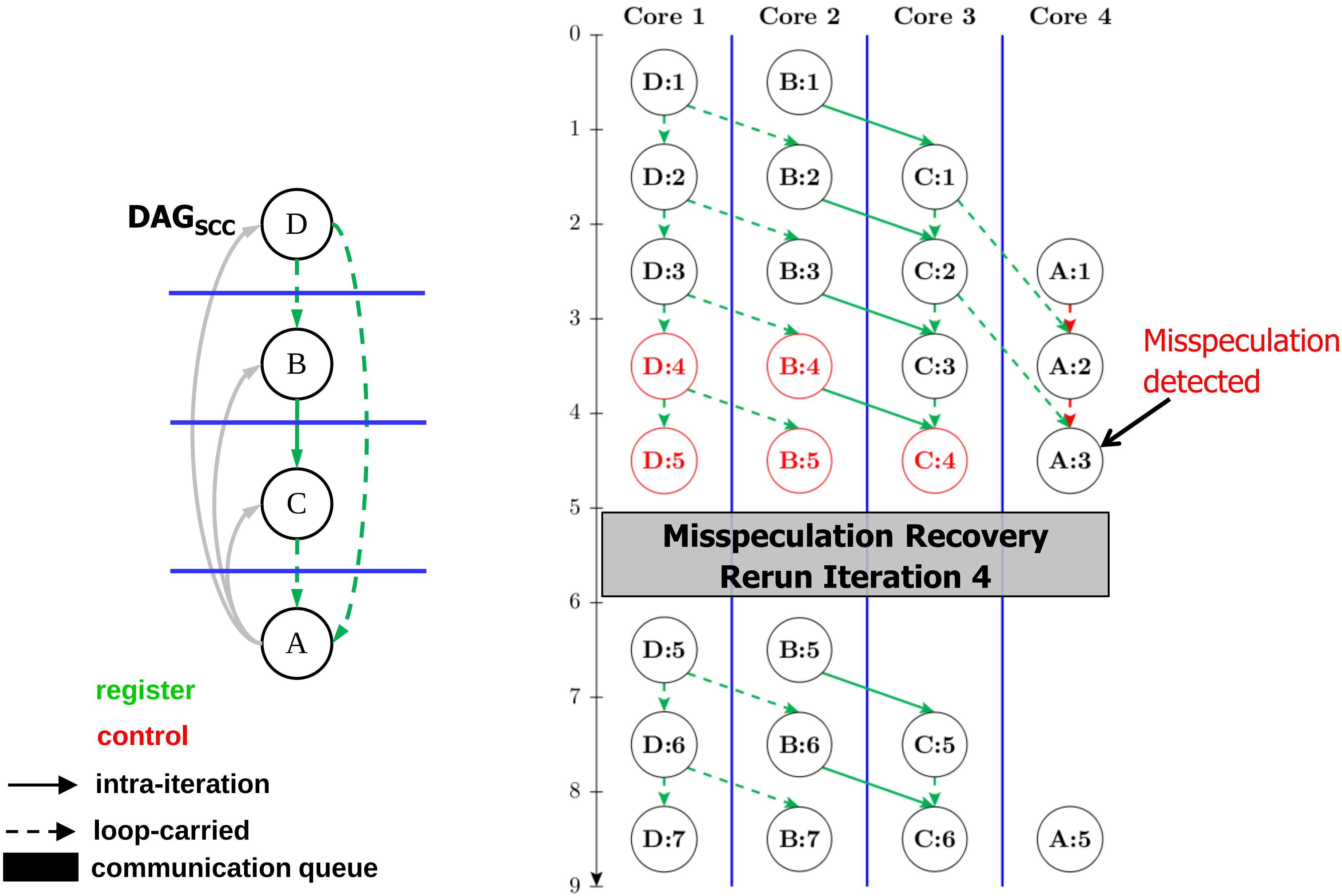


Why Speculation?

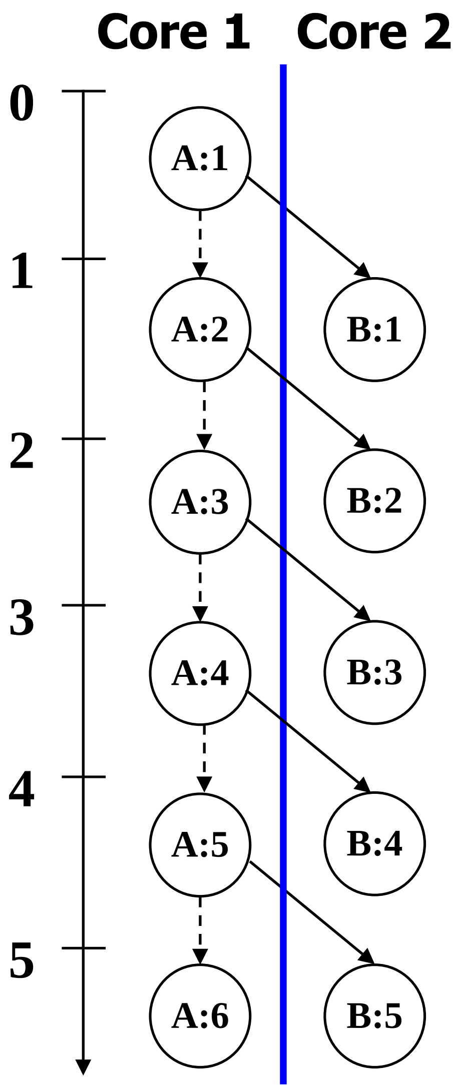
```
A: while(cost < T && node)
B:    ncost = doit(node);
C:    cost += ncost;
D:    node = node->next;
```

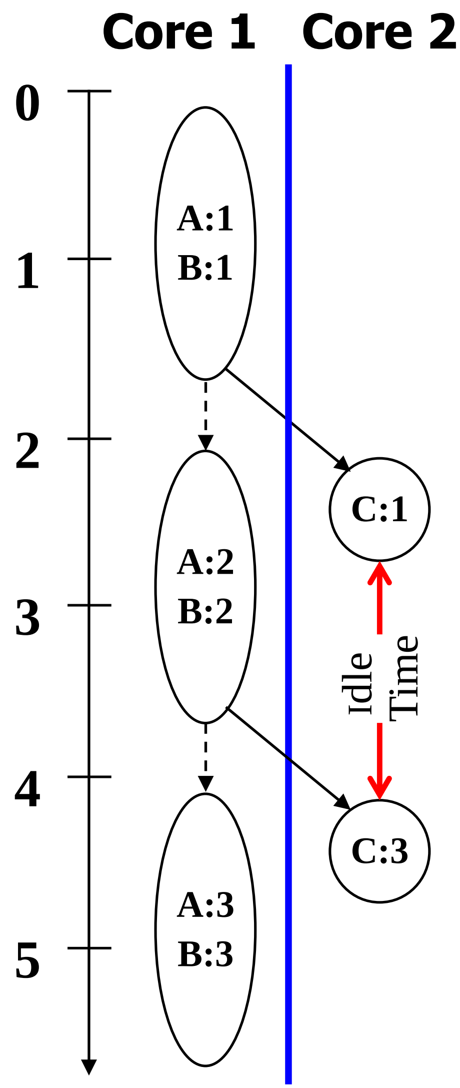


Execution Paradigm



Understanding PMT Performance





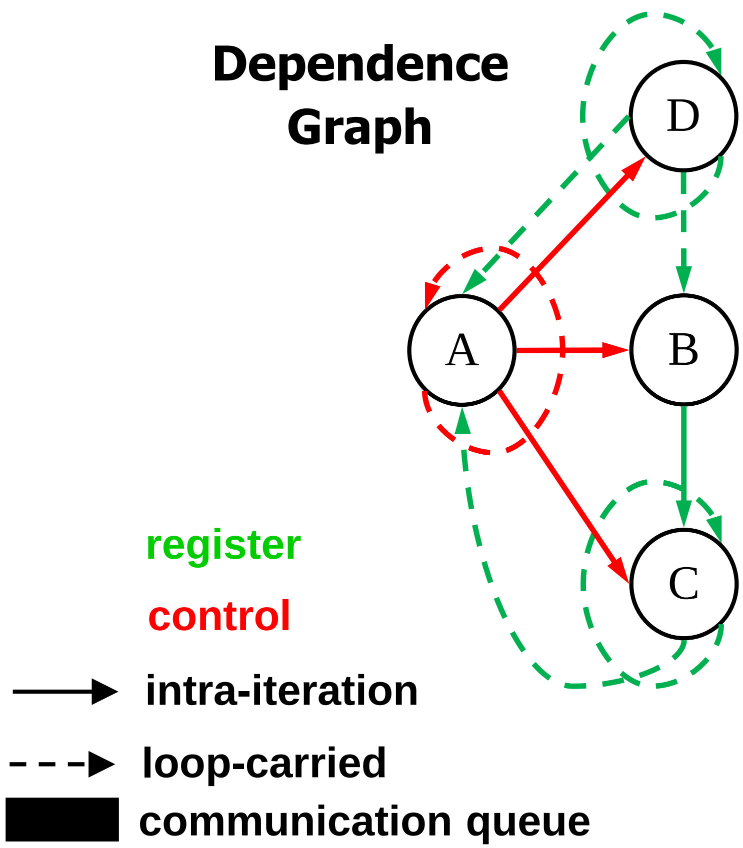
$$T \propto \max(t_i)$$

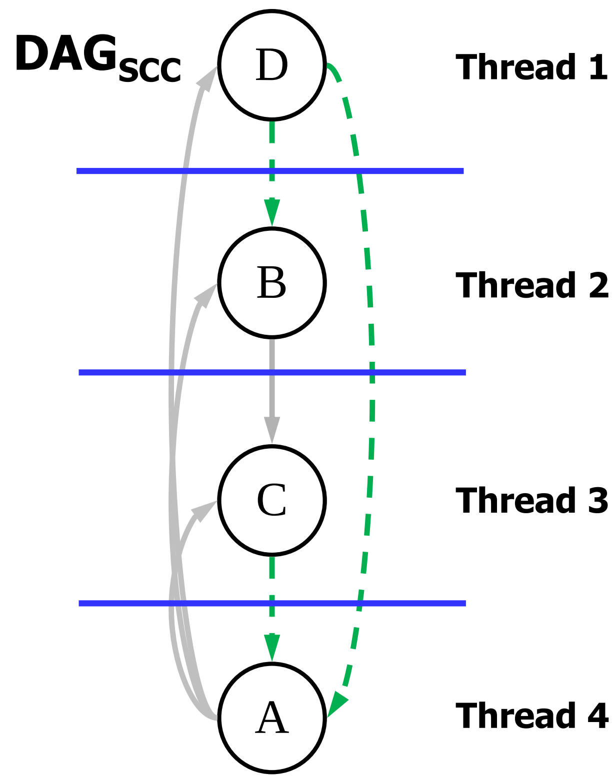
1. **Rate t_i is at least as large as the longest dependence recurrence.**
2. **NP-hard to find longest recurrence.**
3. **Large loops make problem difficult in practice.**

Slowest thread:	1 cycle/iter	2 cycle/iter
Iteration Rate:	1 iter/cycle	0.5 iter/cycle

Selecting Dependences To Speculate

```
A: while(cost < T && node)
B:    ncost = doit(node);
C:    cost += ncost;
D:    node = node->next;
```





Detecting Misspeculation

Thread 1

```
A^1: while(consume(4))
D :    node = node->next
       produce({0,1},node);
```

Thread 2

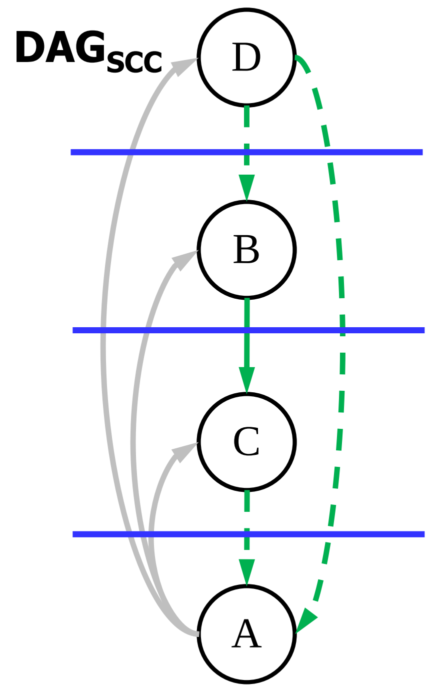
```
A^2: while(consume(5))
B :    ncost = doit(node);
       produce(2,ncost);
D^2:   node = consume(0);
```

Thread 3

```
A^3: while(consume(6))
B^3:   ncost = consume(2);
C :    cost += ncost;
       produce(3,cost);
```

Thread 4

```
A : while(cost < T && node)
B^4:   cost = consume(3);
C^4:   node = consume(1);
       produce({4,5,6},cost < T
                       && node);
```



Detecting Misspeculation

Thread 1

```
A^1: while(TRUE)
D  :   node = node->next
       produce({0,1},node);
```

Thread 2

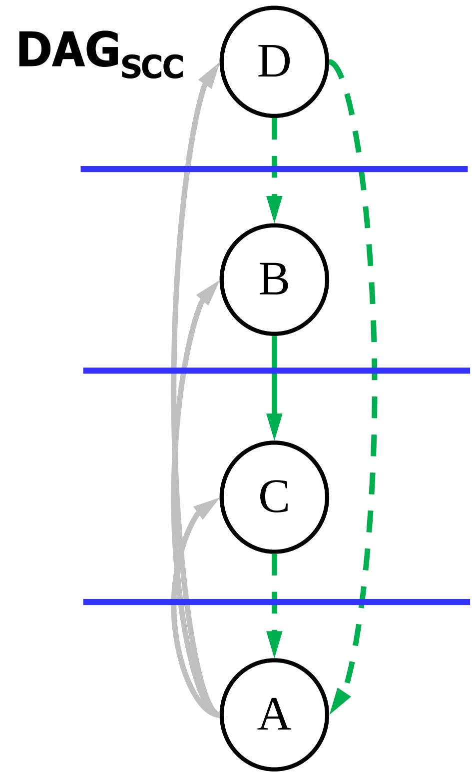
```
A^2: while(TRUE)
B  :   ncost = doit(node);
       produce(2,ncost);
D^2:   node = consume(0);
```

Thread 3

```
A^3: while(TRUE)
B^3:   ncost = consume(2);
C  :   cost += ncost;
       produce(3,cost);
```

Thread 4

```
A  : while(cost < T && node)
B^4:   cost = consume(3);
C^4:   node = consume(1);
       produce({4,5,6},cost < T
                   && node);
```



Detecting Misspeculation

Thread 1

```
A1: while(TRUE)
D :   node = node->next
      produce({0,1},node);
```

Thread 2

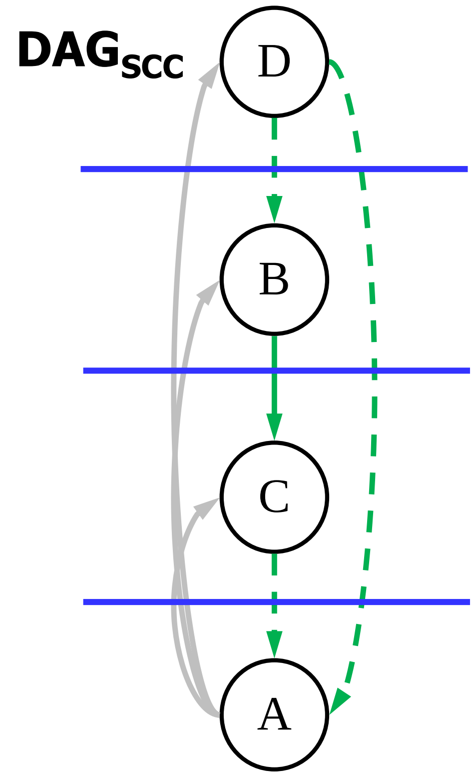
```
A2: while(TRUE)
B :   ncost = doit(node);
      produce(2,ncost);
D2:   node = consume(0);
```

Thread 3

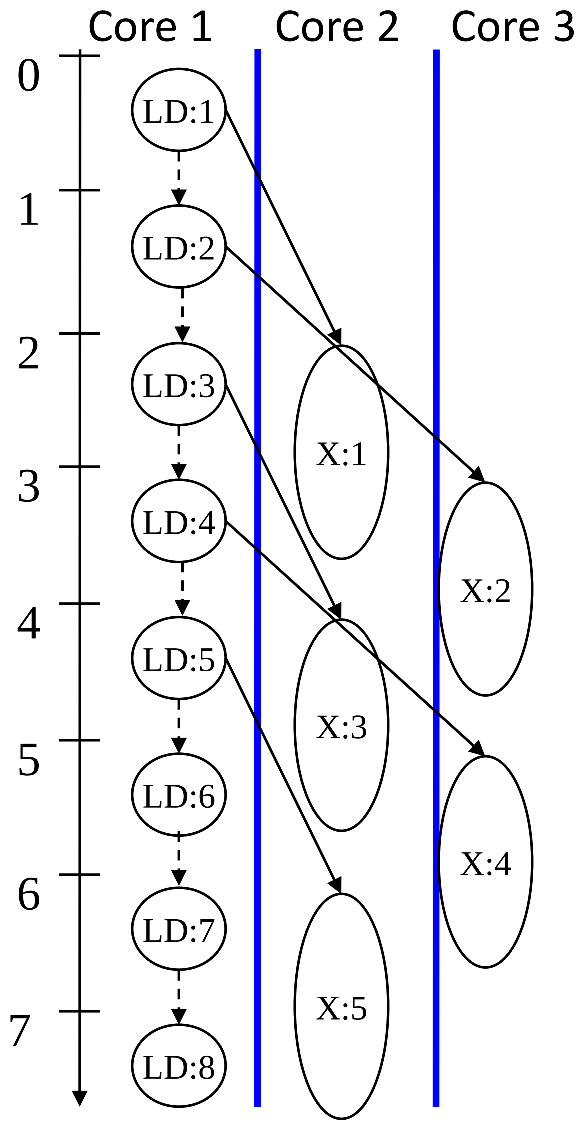
```
A3: while(TRUE)
B3:   ncost = consume(2);
C :   cost += ncost;
      produce(3,cost);
```

Thread 4

```
A : while(cost < T && node)
B4:   cost = consume(3);
C4:   node = consume(1);
      if(!(cost < T && node))
        FLAG_MISSPEC();
```



Adding Parallel Stages to DSWP




```
while(ptr = ptr->next)      // LD
  ptr->val = ptr->val + 1; // X
```

LD = 1 cycle
X = 2 cycles

Comm. Latency = 2 cycles

Throughput
DSWP: 1/2 iteration/cycle
DOACROSS: 1/2 iteration/cycle
PS-DSWP: 1 iteration/cycle

Things to Think About

- How do you decide what dependences to speculate?
  - » Look solely at profile data?
  - » How do you ensure enough profile coverage?
  - » What about code structure?
  - » What if you are wrong? Undo speculation decisions at run-time?
- How do you manage speculation in a pipeline?
  - » Traditional definition of a transaction is broken
  - » Transaction execution spread out across multiple cores
- How many cores can DSWP realistically scale to?
  - » Can a pipeline be adjusted when the number of available cores increases/decreases, or based on what else is running on the processor?