WISCONSIN

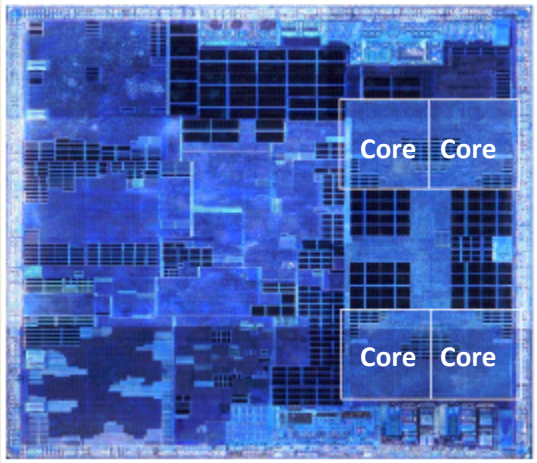
UNIVERSITY OF WISCONSIN–MADISON

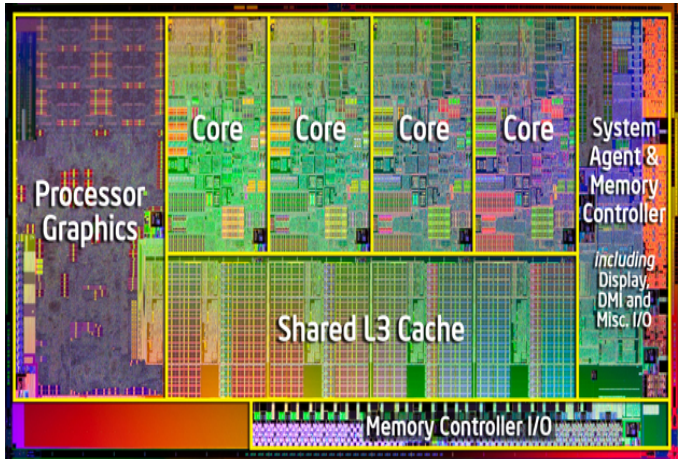
Dataflow Execution of Sequential Imperative Programs on Multicore Architectures

Gagan Gupta

Gurindar S. Sohi

Multicores in General Purpose Systems





Programming Multicores

- Statically-parallel programs
- Explicit parallel execution
- Correctness of parallel execution?

Programming Multicores

Solving Challenges: Current Approach

- Fix the shortcomings
- User Driven
  - Prune nondeterminism
  - Recreate original sequence
- Research Proposals
  - Reintroduce repeatability

Modern Programming Practices

- Object-oriented programming
- Reusable modular design
- Data encapsulation
- Imperative languages

Execution Models

- Dataflow execution
  - Expose innate parallelism
- Superscalar processors
  - Dynamic parallelism
  - Statically-sequential programs

Executive Summary

?

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Executive Summary

Statically-Sequential Programs

Dataflow Parallel Execution

Outline

- Proposed Model
- Prototype
- Evaluation

Outline

- Proposed Model

- Prototype
- Evaluation

Developing the Model for Multicores

- Sequential imperative programming language
  - Leverage modern programming principles
  - Provision for worst-case scenarios

- Achieving dataflow parallel execution

Program Decomposition

Static Program

```
object A
..

while (cond) {
  function F ( )
}
function F' ( )
```

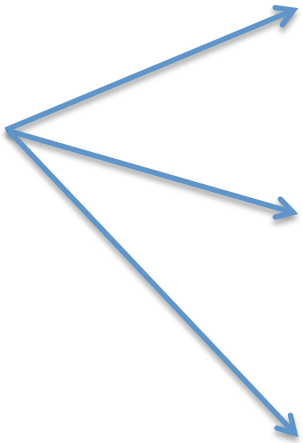
Dynamic Sequence

```
..
evaluate cond (PS1)
F1 ( )
evaluate cond (PS2)
F2 ( )
evaluate cond (PS3)
F3 ( )
..
```

Program Decomposition

Static Program

```
object A
..

while (cond) {
  function F ( )
}
function F' ( )
```



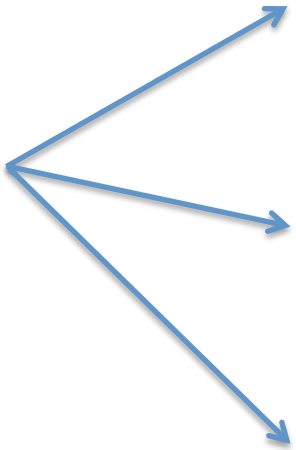
Functions

```
..
evaluate cond (PS1)
F1 ( )
evaluate cond (PS2)
F2 ( )
evaluate cond (PS3)
F3 ( )
..
```

Program Decomposition

Static Program

```
object A
..

while (cond) {
  function F ( )
}
function F' ( )
```

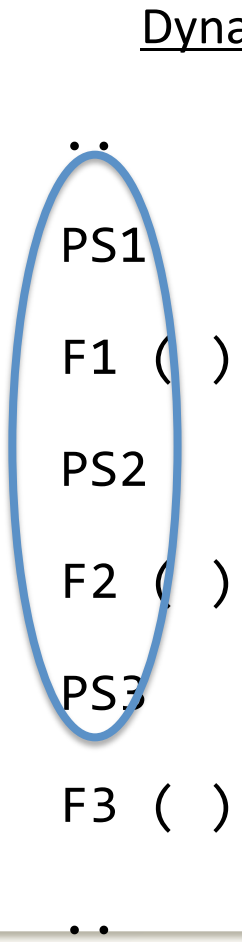


Program Segments

```
..
evaluate cond (PS1)
F1 ( )
evaluate cond (PS2)
F2 ( )
evaluate cond (PS3)
F3 ( )
..
```

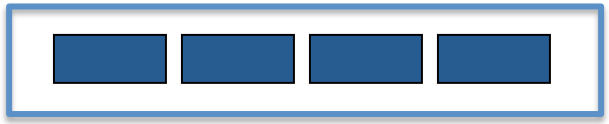
Program Decomposition

- Collection of computations
- Implicit order



Program Execution

Multicore Processor

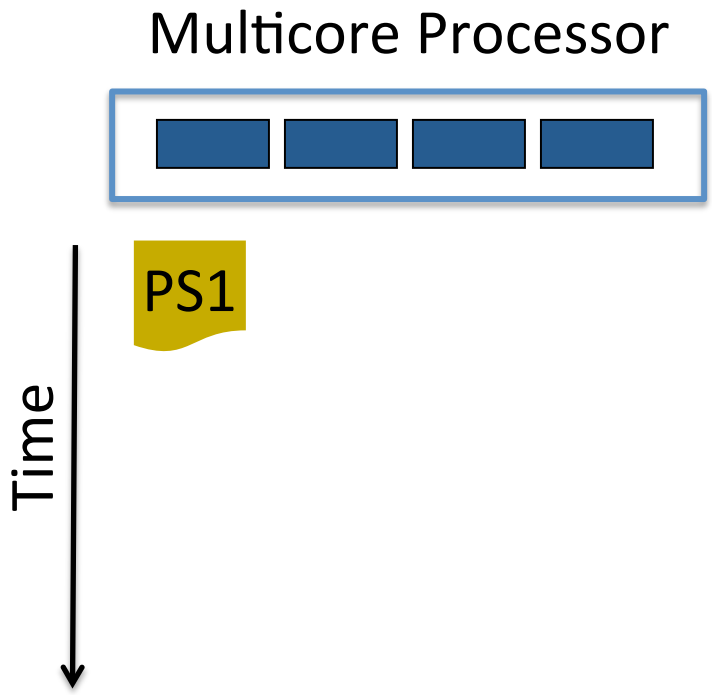


Time

Dynamic Sequence

```
..
PS1
F1 ( )
PS2
F2 ( )
PS3
F3 ( )
..
```

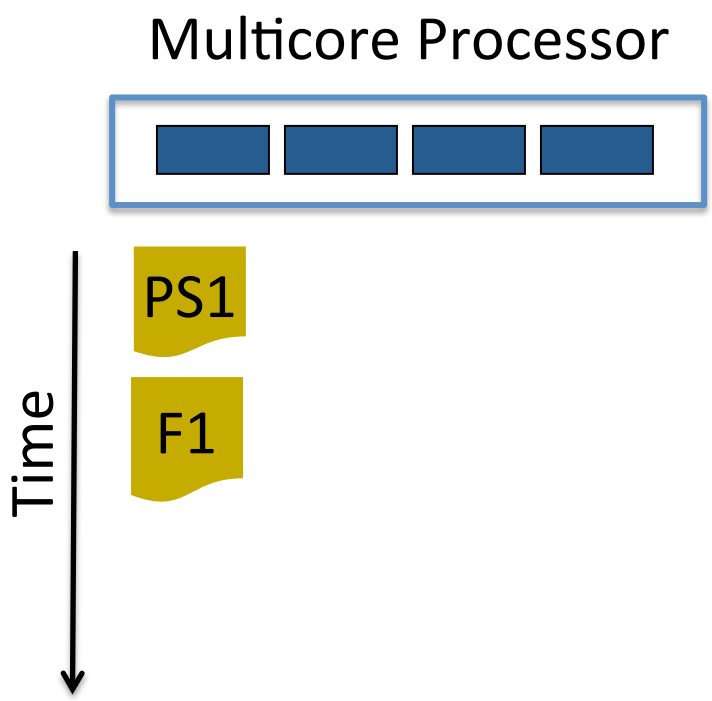
Program Execution



Dynamic Sequence

```
..
PS1
F1 ( )
PS2
F2 ( )
PS3
F3 ( )
..
```

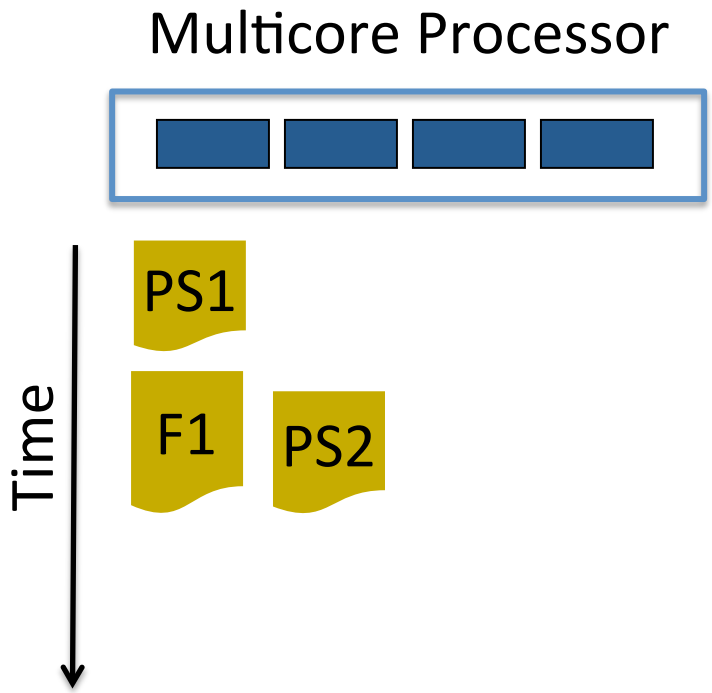
Program Execution



Dynamic Sequence

```
..
PS1
F1 ( )
PS2
F2 ( )
PS3
F3 ( )
..
```

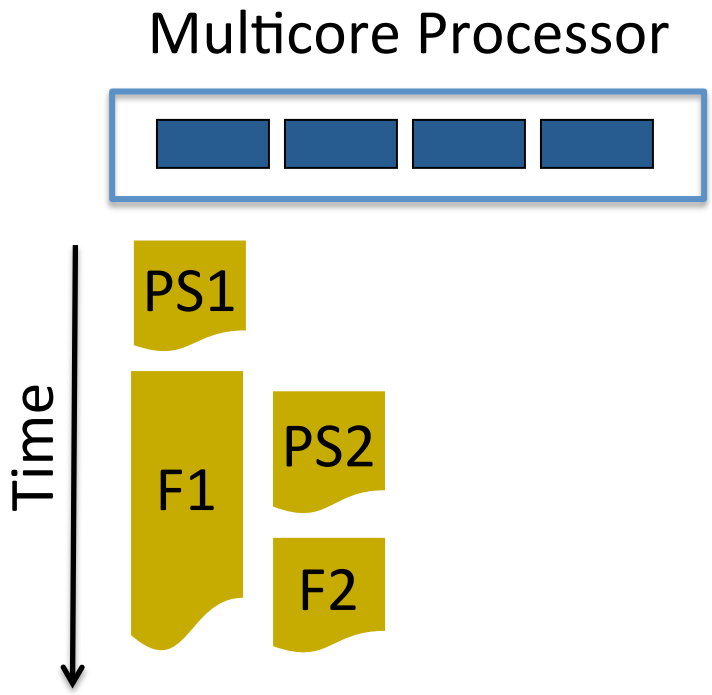
Program Execution



Dynamic Sequence

```
..
PS1
F1 ( )
PS2
F2 ( )
PS3
F3 ( )
..
```

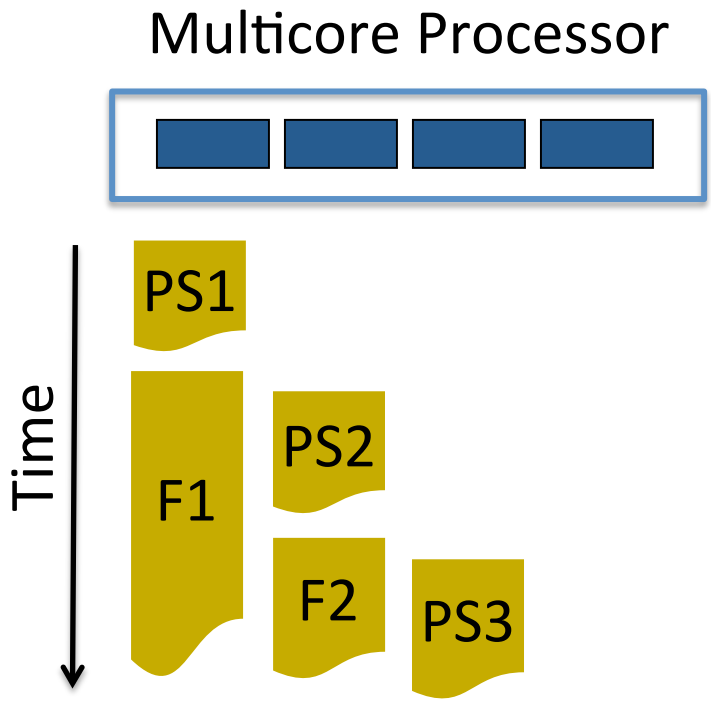
Program Execution



Dynamic Sequence

```
..
PS1
F1 ( )
PS2
F2 ( )
PS3
F3 ( )
..
```

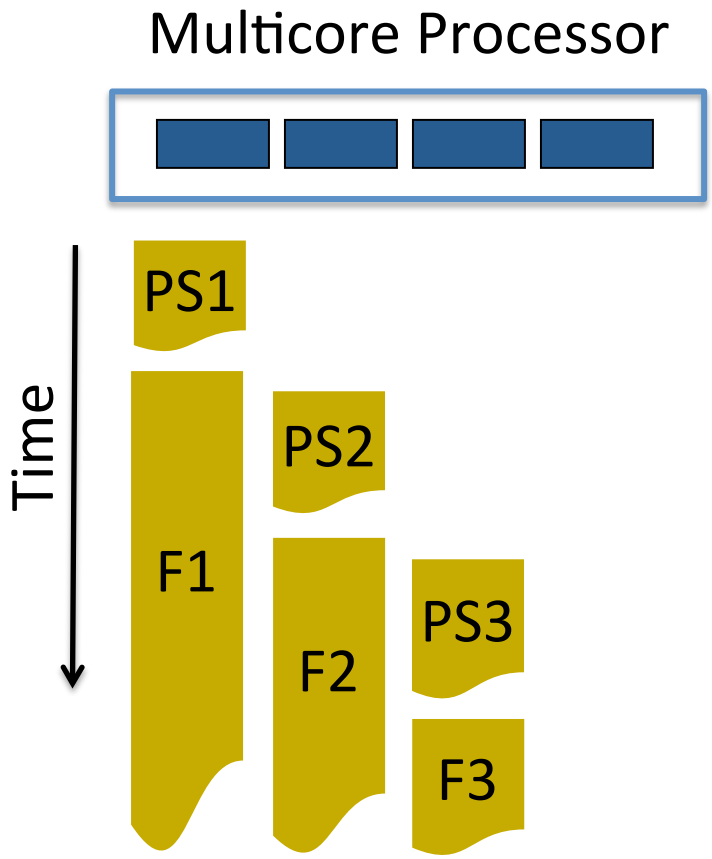
Program Execution



Dynamic Sequence

```
..
PS1
F1 ( )
PS2
F2 ( )
PS3
F3 ( )
..
```

Program Execution



Dynamic Sequence

..

PS1

F1 ( )

PS2

F2 ( )

PS3

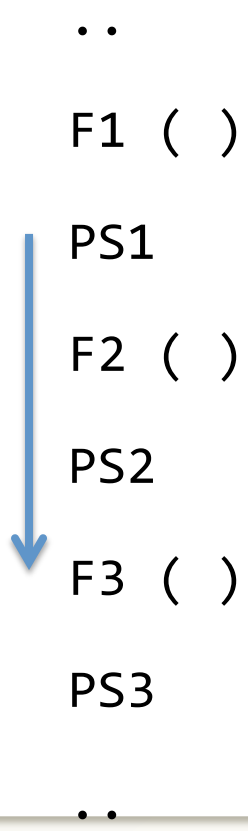
F3 ( )

..

Program Execution

- Implicit order
  - \- Hindrance
  - \+ Exploitable

Dynamic Sequence



Level of Abstraction

```
object A
..

while (cond) {
  function F ( )
}
function F' ( )
```

"Instruction"

Level of Abstraction

```
object A
..

while (cond) {
  function F ( ..interface.. )
}
function F' ( )
```

"Operands"

Level of Abstraction

```
object A
..

while (cond) {
  function F ({wr_set} {rd_set})
}
function F' ( )
```

"Operands"

Level of Abstraction

object A

"Unit of Data"

```
..

while (cond) {
  function F ({wr_set} {rd_set})
}
function F' ( )
```

Achieving Dataflow Parallel Execution

- Our view of a program
  - Computations
  - Level of abstraction
- Classical dataflow machines
  - Tokens: data dependence
  - Resource management

Achieving Dataflow Parallel Execution

- Proposed Model
  - Level of abstraction
  - Imperative programs can mutate data
- Token protocol
  - Data dependence
- Program order

Data Dependence

```
object A
..

while (cond) {
  function F ({wr_set} {rd_set})
}
function F' ( )
```

Data Dependence: Identifying Objects

```
object A
..

while (cond) {
  function F ({wr_set} {rd_set})
}
function F' ( )
```

```
F1 {B, C} {A}

F2 {D} {A}

F3 {A, E} {I}

F4 {B} {D}

F5 {B} {D}

F6 {G} {H}
```

Data Dependence: Unknown Data Set

```
object A
..

while (cond) {
  function F ({wr_set} {rd_set})
}
function F' ({?} {?})
```

```
F1 {B, C} {A}

F2 {D} {A}

F3 {A, E} {I}

F4 {B} {D}

F5 {B} {D}

F6 {G} {H}

F' {?} {?}
```

Data Dependence

F1 {B, C} {A}

F2 {D} {A}

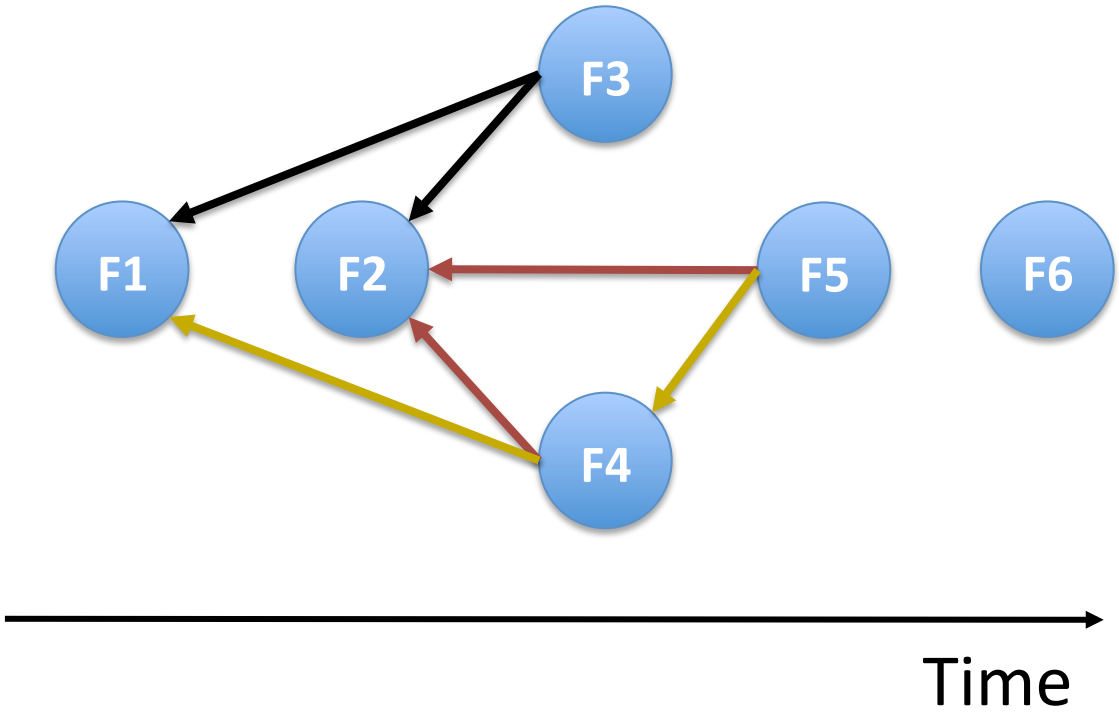
F3 {A, E} {I}

F4 {B} {D}

F5 {B} {D}

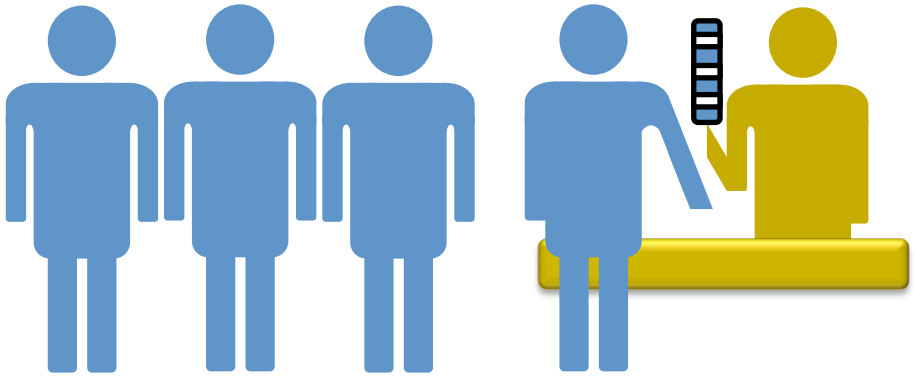
F6 {G} {H}

F’ {?} {?}



Token Protocol

object A



Token Protocol: Acquire

B

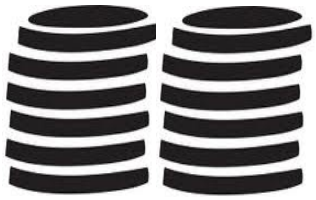
A

F1

P

Token Protocol: Acquire

B



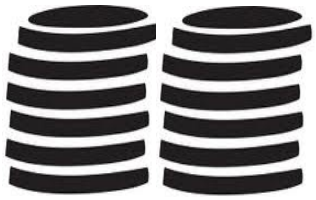
A

F1

P

Token Protocol: Submit

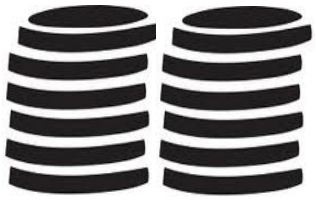
B



A

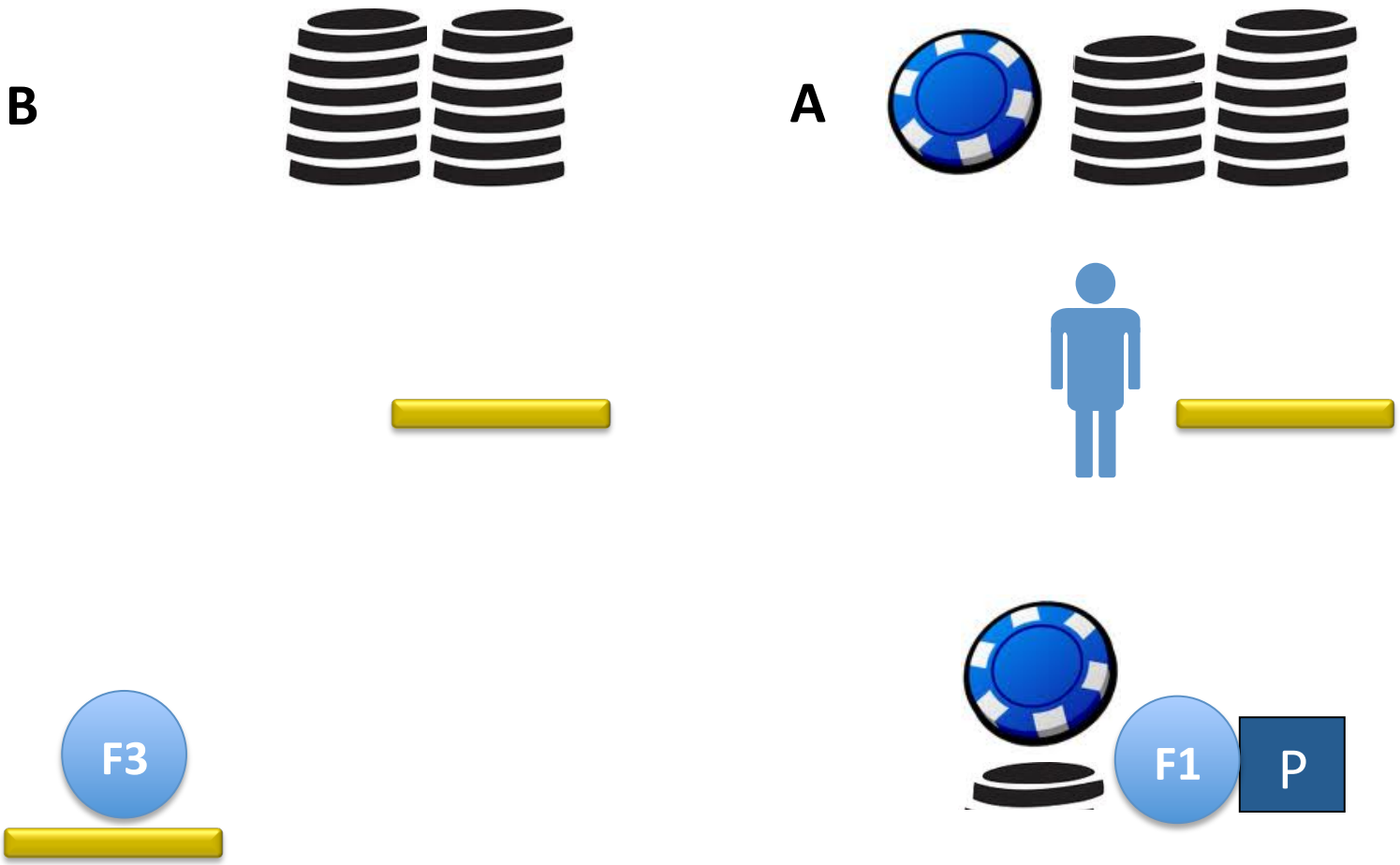
Token Protocol: Shelve

B

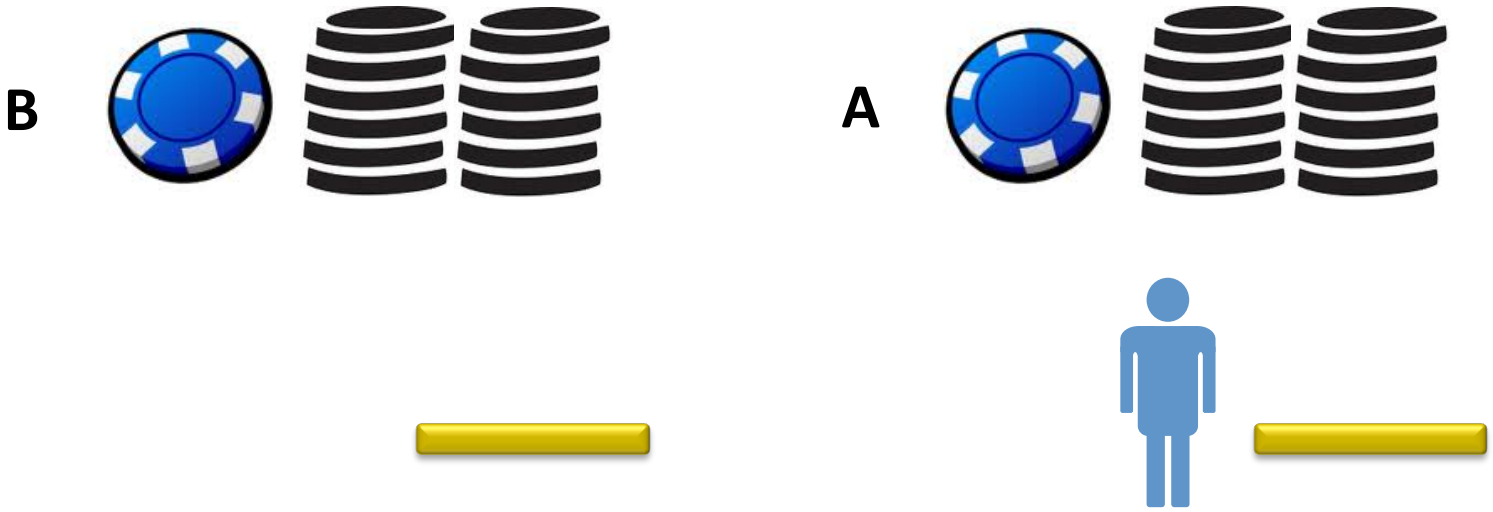


A

Token Protocol: Shelve

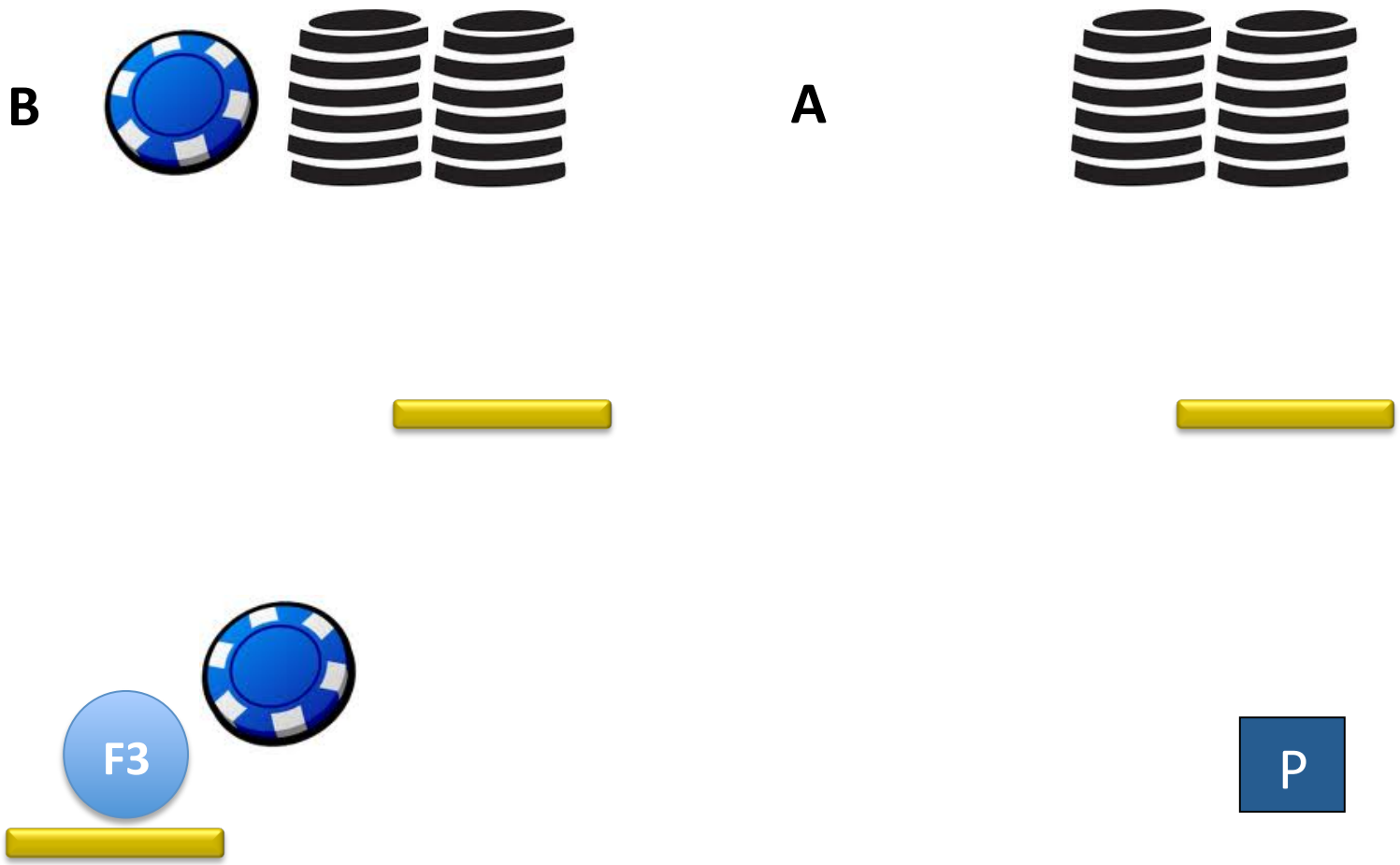


Token Protocol: Release



Token Protocol: Pass

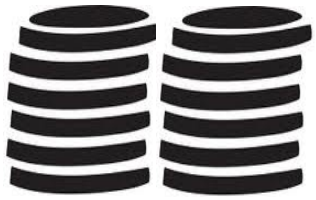
Token Protocol: Schedule



Token Protocol: Schedule

B

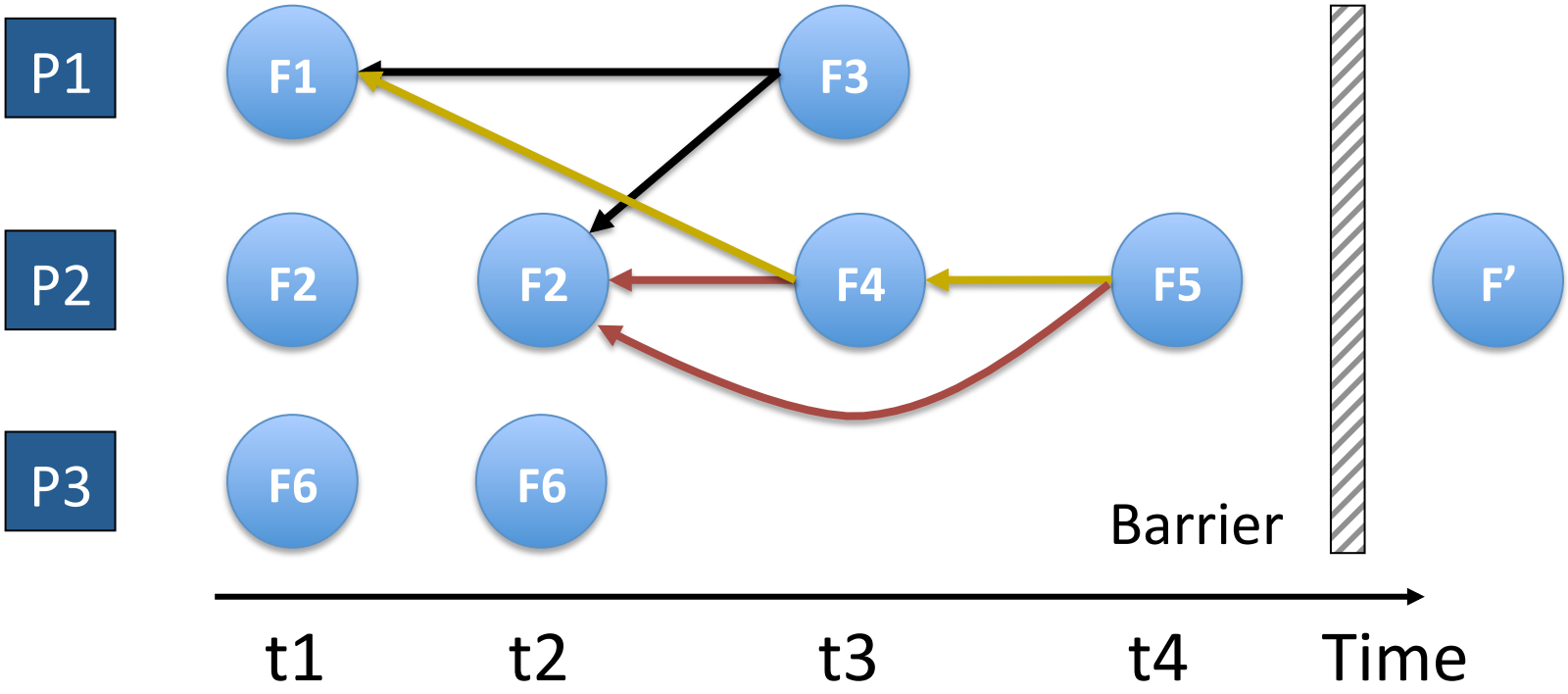
A



Exploiting Program Order

- Deadlock avoidance in token protocol
  - Process functions in program order
  - Grant tokens in request order
- Resource management
  - Unravel as much as resources allow
  - Guarantee forward progress

Dataflow Execution on Multicores



Proposed Model: Summary

- Sequentially determinate, race-free
  - Repeatable
  - Predictable
- Dataflow execution
  - No explicit parallelization
- ~~Correctness of parallel execution?~~

Outline

- Proposed Model
- Prototype
- Evaluation

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Prototype

- Runtime library (C++ currently)
- Implements token protocol
- Achieves parallel execution

Composing Programs

Sequential Program

```
Object A
..

while (cond) {
  function F: ( .. )
}

Function F': {?} {?}
```

Program in Proposed Model

```
Object A
..

while (cond) {

}


F'( .. );
```

Composing Programs

Sequential Program

```
Object A
..

while (cond) {
  function F: ( .. )
}

Function F': {?} {?}
```

Program in Proposed Model

```
Object A : Token
..

while (cond) {

}


F'( .. );
```

Composing Programs

Sequential Program

```
Object A
..

while (cond) {
  function F: ( .. )
}

Function F’: {?} {?}
```

Program in Proposed Model

```
Object A : Token
..

while (cond) {
  df_execute (                &F);
}


F’( .. );
```

Composing Programs

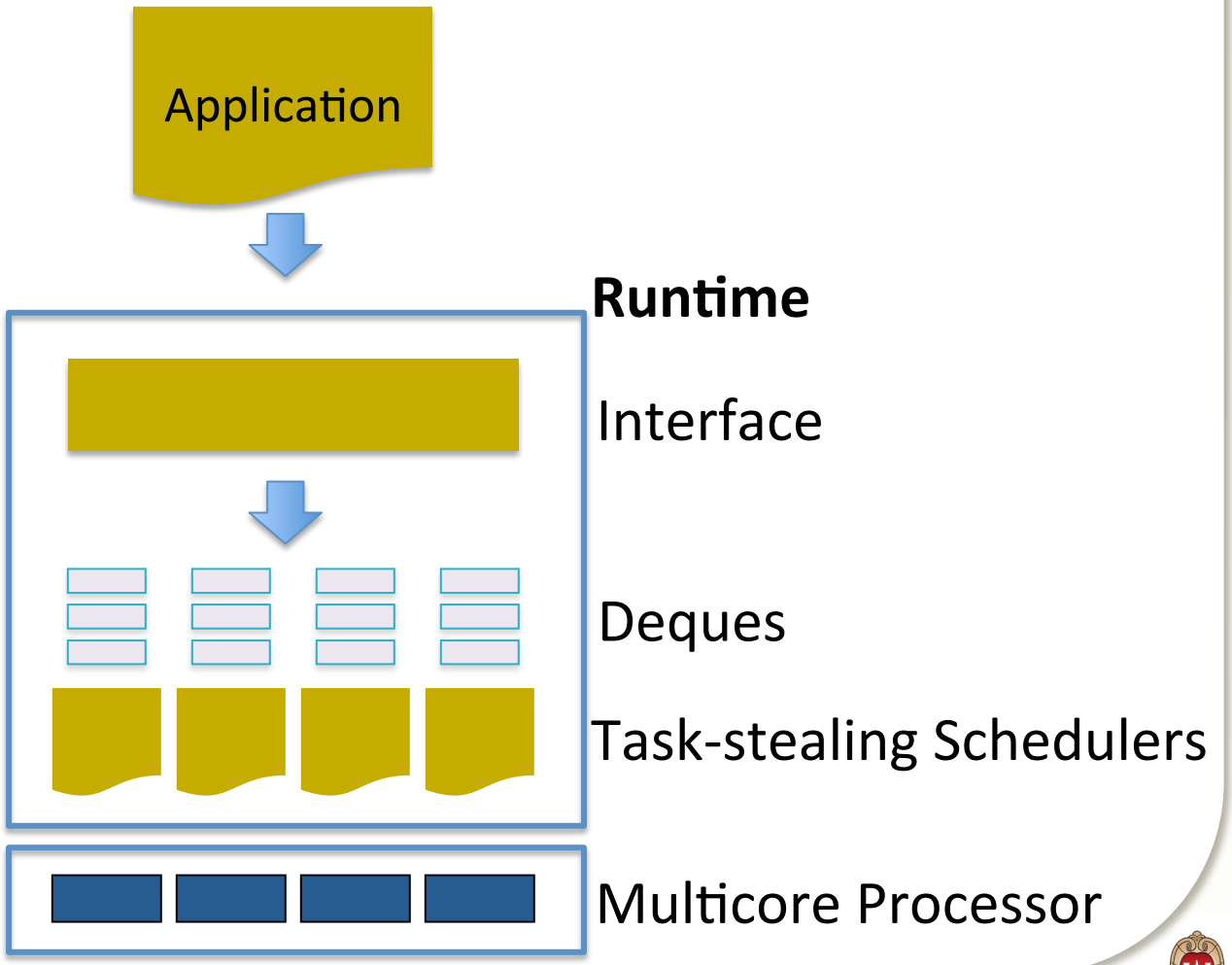
Sequential Program

```
Object A
..

while (cond) {
  function F: ( .. )
}

Function F': {?} {?}
```

Program in Proposed Model

```
Object A : Token
..

while (cond) {
  df_execute (wr_set, rd_set, &F);
}

df_end ( );
F'( .. );
```

Parallel Execution Mechanics



Outline

- Proposed Model
- Prototype
- Evaluation

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Evaluation

- Ease of programming (qualitative)
- Performance & overheads
- Setup
  - Microbenchmarks & benchmarks
  - 3 stock multicore machines

bzip2

Sequential

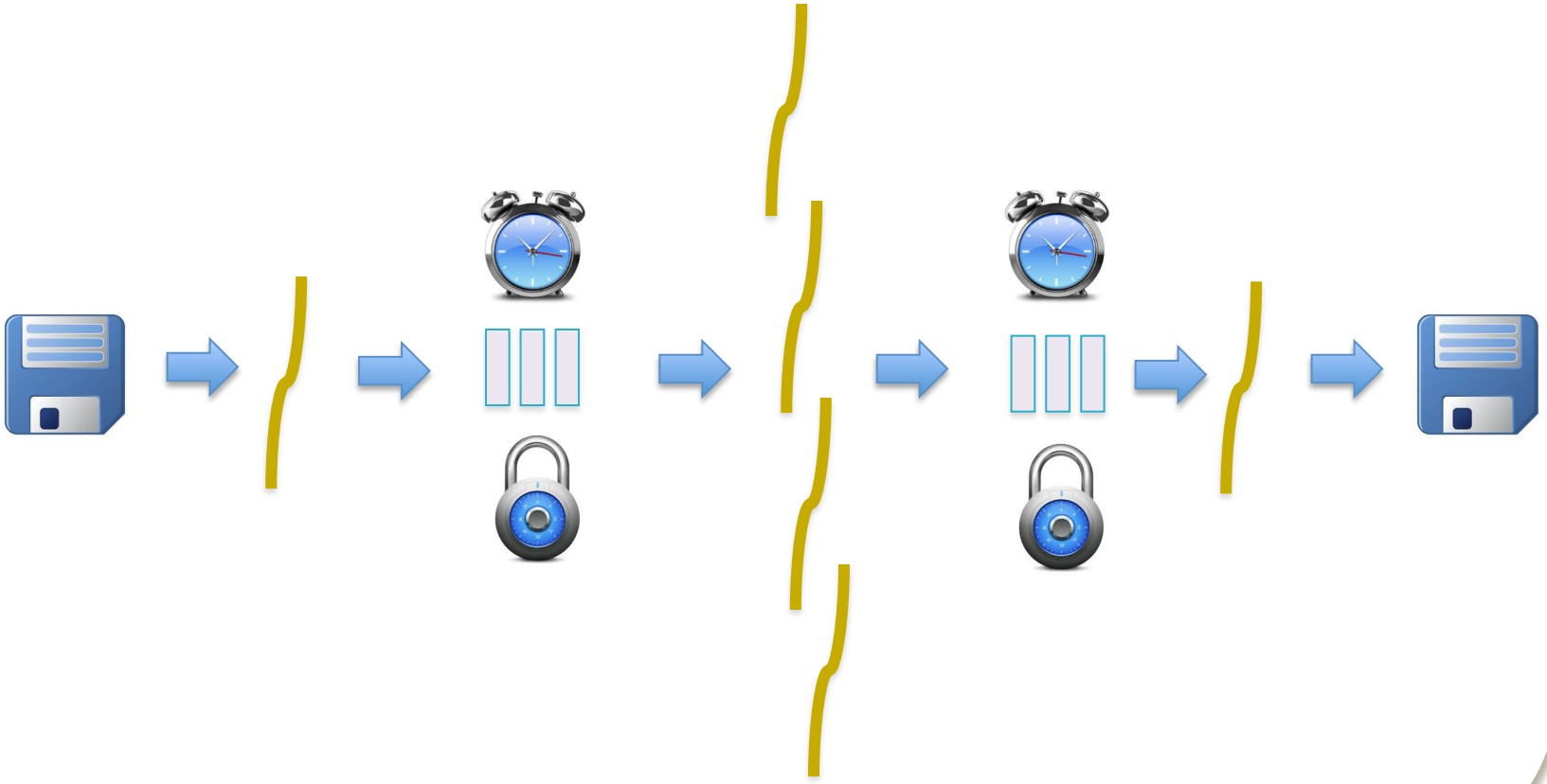
```
..
..
while (!EOF) {

    block = new block_t (InFile, size);

    compress (block)
    file_write (OpFile, block);
}
..
```

bzip2: Pthread

```
 for (;;)       {
    pthread_mutex_lock(fifo->mut);
    while (fifo->empty)             {
    if (allDone == 1)             {
      pthread_mutex_unlock(fifo->mut);
      return (NULL);
    }
    GetSystemTime(&systemtime);
    SystemTimeToFileTime(&systemtime, (FILETIME
*)&filetime);
    waitTimer.tv_sec = filetime.QuadPart / 10000000;
    waitTimer.tv_nsec = filetime.QuadPart -
((LONGLONG)waitTimer.tv_sec * 10000000) * 10;
    waitTimer.tv_sec++;
    pret = pthread_cond_timedwait(fifo->notEmpty,
fifo->mut, &waitTimer);
    FileData = queueDel(fifo, &inSize, &blockNum);
    pthread_mutex_unlock(fifo->mut);
    pret = pthread_cond_signal(fifo->notFull);
    outSize = (int) ((inSize*1.01)+600);
    pthread_mutex_lock(MemMutex);
   CompressedData = new char[outSize];
   pthread_mutex_unlock(MemMutex)
   if (CompressedData == NULL) {
```

```
 while ((currBlock < NumBlocks) || (allDone == 0)) {
    pthread_mutex_lock(OutMutex);
    if ((OutputBuffer.size() == 0) || (OutputBuffer
[currBlock].bufSize < 1) || (OutputBuffer[currBlock].buf
== NULL))    {
       pthread_mutex_unlock(OutMutex);
       usleep(50000);
       continue;
      } else
       pthread_mutex_unlock(OutMutex);

ret = write(hOutfile, OutputBuffer[currBlock].buf,
OutputBuffer[currBlock].bufSize);
CompressedSize += ret;
 pthread_mutex_lock(OutMutex);

pthread_mutex_lock(MemMutex);
if (OutputBuffer[currBlock].buf != NULL)   {
   delete [] OutputBuffer[currBlock].buf;
   NumBufferedBlocks--;
}
pthread_mutex_unlock(MemMutex);
pthread_mutex_unlock(OutMutex);
currBlock++;
```

bzip2: Pthread



bzip2: Proposed Model

Sequential

```
..
..
while (!EOF) {

    block = new block_t (InFile, size);

    compress (block)
    file_write (OpFile, block);
}
..
```

Proposed Model

```
..

while (!EOF) {

    block = new block_t (InFile, size);

}
..
```

bzip2: Proposed Model

Sequential

```
..
..
while (!EOF) {

    block = new block_t (InFile, size);

    compress (block)
    file_write (OpFile, block);
}
..
```

Proposed Model

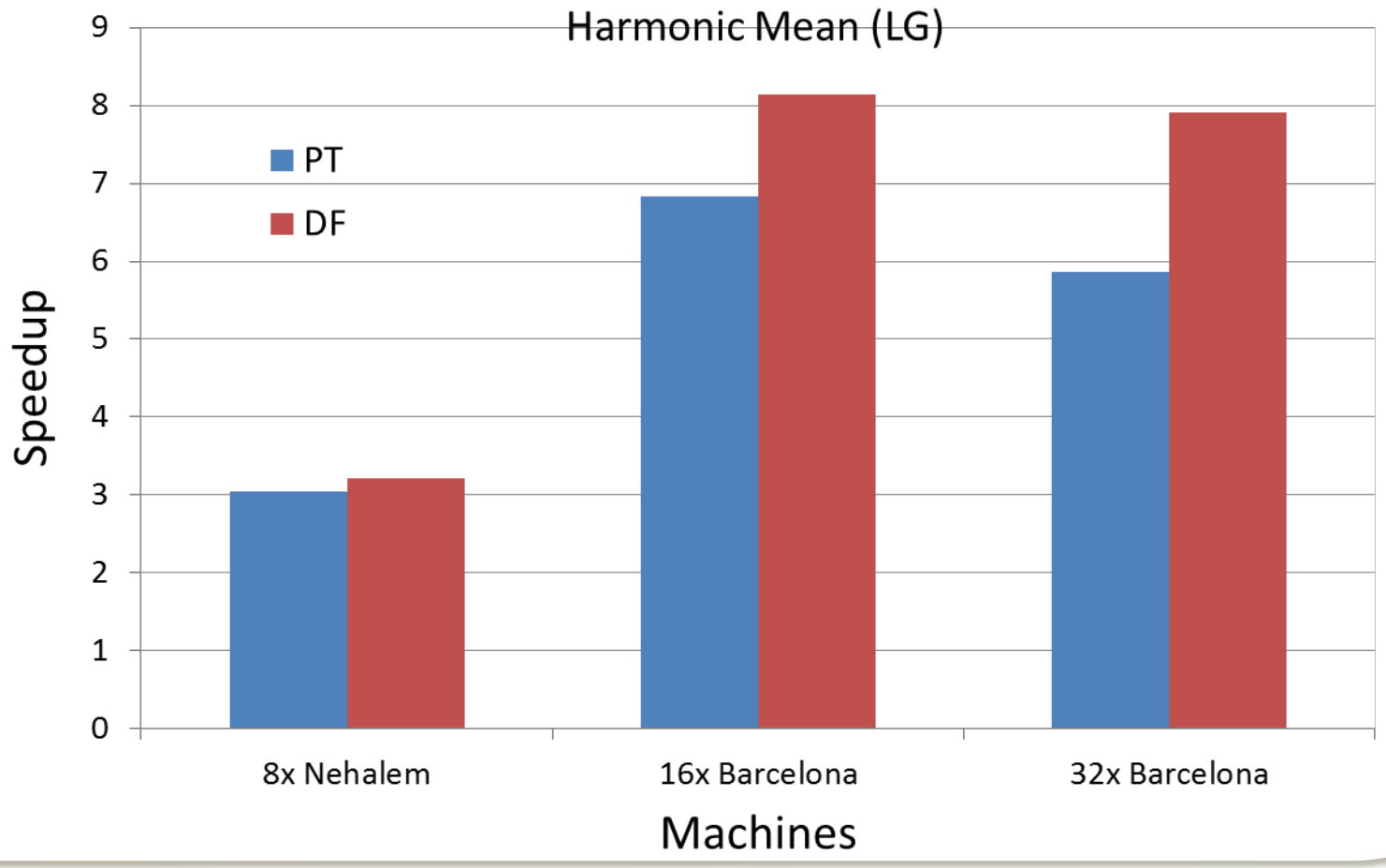
```
..
opset.insert (OpFile);
while (!EOF) {

    block = new block_t (InFile, size);
    blset.insert (block);
    df_execute (blset,  &compress);
    df_execute (opset, blset, &file_write);
}
..
```

Results

Benchmark
barneshut
barneshut-LG
blackscholes
blackscholes-LG
bzip2
bzip2-LG
dedup
histogram
reverse_index
microbenchmarks

- Machines
  - 4-core/8-thread Nehalem Core i7
  - 16-core AMD Opteron 8350
  - 32-core AMD Opteron 8356
- Input sizes
  - Small, Medium, Large

Results: Competitive Performance



Other Results

- Performance (benchmarks)
  - Different task granularities
  - Scalability
- Overheads (benchmarks & microbenchmarks)
  - Token protocol
  - Granularity requirements
- Resource usage (benchmarks & microbenchmarks)

Related Work

- Determinate models
  - Jade, Serialization Sets, DPJ
  - SMPSs, Task Superscalar
- Nondeterministic models
  - Cilk, TBB

Summary

- Novel execution model for multicores
  - Statically-Sequential Imperative Programs
  - Dataflow Parallel Execution
- Benefits
  - Sequentially Determinate Execution
  - Competitive Performance

Related Work: Determinate Models

- SMPSs (StarSs)
  - Task-flow graph based on memory locations
  - Data renaming
  - Centralized scheduler
- Task Superscalar
  - Emulates superscalar processors
- DPJ
  - Static, type-and-effect system

Related Work: Nondeterministic Models

- Task-based
  - Cilk, TBB
- Multithreaded
  - Pthread

Overheads

Overhead	Core i7 (cycles)	AMD 8350 (cycles)	AMD 8356 (cycles)
Token Processing (Base)	Few 100 to Few 1000		
Additional object	Few 100 to Few 1000		
Granularity (Base)	4000	12000	18000
Additional object	2000	2000	4000