Introduction to Cilk

Programming Models for Emerging Platforms

First Model: POSIX Threads

- C library for thread programming
- Almost as crude as it gets

What does this mean?

First Model: POSIX Threads

- With threads, programmers must:
  - 1. Manage physical unit of execution (thread)
  - 2. Manage logical unit of execution (code)
  - 3. Synchronize these physical and logical units of execution
  - 4. Handle life cycle dependency *between* threads
  - 5. Handle memory dependency *between* threads
  - 6. Deal with challenging side effect of concurrency: race conditions and deadlock

First Model: POSIX Threads

- With threads, programmers must:
  - 1. Manage physical unit of execution (thread)
  - 2. Manage logical unit of execution (code)
  - 3. Synchronize these physical and logical units of execution
  - 4. Handle life cycle dependency *between* threads
  - 5. Handle memory dependency *between* threads
  - 6. Deal with challenging side effect of concurrency: race conditions and deadlock

First Model: POSIX Threads

- With threads, programmers must:
  - 1. Manage physical unit of execution (thread)
  - 2. Manage logical unit of execution (code)
  - 3. Synchronize these physical and logical units of execution
  - 4. Handle life cycle dependency *between* threads
  - 5. Handle memory dependency *between* threads
  - 6. Deal with challenging side effect of concurrency: race conditions and deadlock

First Model: POSIX Threads

- With threads, programmers must:
  - 1. Manage physical unit of execution (thread)
  - 2. Manage logical unit of execution (code)
  - 3. Synchronize these physical and logical units of execution
  - 4. Handle life cycle dependency *between* threads
  - 5. Handle memory dependency *between* threads
  - 6. Deal with challenging side effect of concurrency: race conditions and deadlock

First Model: POSIX Threads

- With threads, programmers must:
  - 1. Manage physical unit of execution (thread)
  - 2. Manage logical unit of execution (code)
  - 3. Synchronize these physical and logical units of execution
  - 4. Handle life cycle dependency *between* threads
  - 5. Handle memory dependency *between* threads
  - 6. Deal with challenging side effect of concurrency: race conditions and deadlock

First Model: POSIX Threads

- With threads, programmers must:
  - 1. Manage physical unit of execution (thread)
  - 2. Manage logical unit of execution (code)
  - 3. Synchronize these physical and logical units of execution
  - 4. Handle life cycle dependency *between* threads
  - 5. Handle memory dependency *between* threads
  - 6. Deal with challenging side effect of concurrency: race conditions and deadlock

First Model: POSIX Threads

- With threads, programmers must:
  - 1. Manage physical unit of execution (thread)
  - 2. Manage logical unit of execution (code)
  - 3. Synchronize these physical and logical units of execution
  - 4. Handle life cycle dependency *between* threads
  - 5. Handle memory dependency *between* threads
  - 6. Deal with challenging side effect of concurrency: race conditions and deadlock

“A bad day writing code in Scheme is better than a good day writing code in C. “

- David Stigant

“A bad day writing code in Scheme is better than a good day writing code in C. “

- David Stigant

My point: There is more to world than C and “performance”

Programming Languages

- We research programming languages to:
  - Provide stronger guarantees (garbage collection = memory safety)
  - Make writing software more productive (spending hours / days / months searching for memory / concurrency bugs is not)
  - Make writing software more *enjoyable*
- A programming model is a subset of programming languages

First Model: POSIX Threads

- With threads, programmers must:
  - 1. Manage physical unit of execution (thread)
  -
  -
  -
  - 5. Handle memory dependency *between* threads
  - 6. Deal with challenging side effect of concurrency: race conditions and deadlock

It’s unlikely we remove all items on this list,
But we can at least shrink it

First Model: POSIX Threads

- With threads, programmers must:
  - 1. Manage physical unit of execution (thread)
  - 2. Manage logical unit of execution (code)
  - 3. Synchronize these physical and logical units of execution
  - 4. Handle life cycle dependency *between* threads
  - 5. Handle memory dependency *between* threads
  - 6. Deal with challenging side effect of concurrency: race conditions and deadlock

Second Model: Cilk

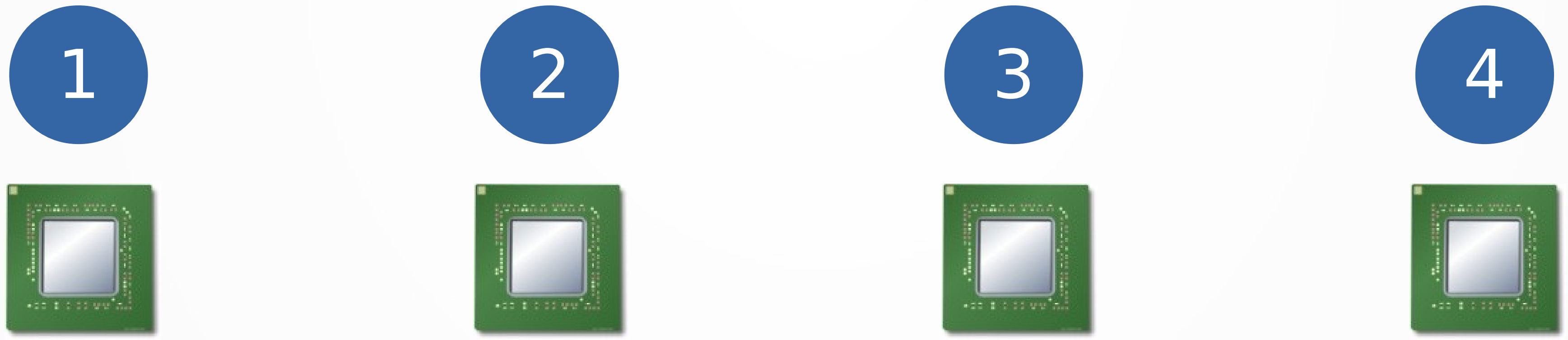
- C language extension for writing parallel and concurrent programs
- Developed at MIT in late 90s
- Part of gcc toolchain (search for intel cilkplus)
- No more threads! We use workers (which are invisible to the programmer) and spawn work.

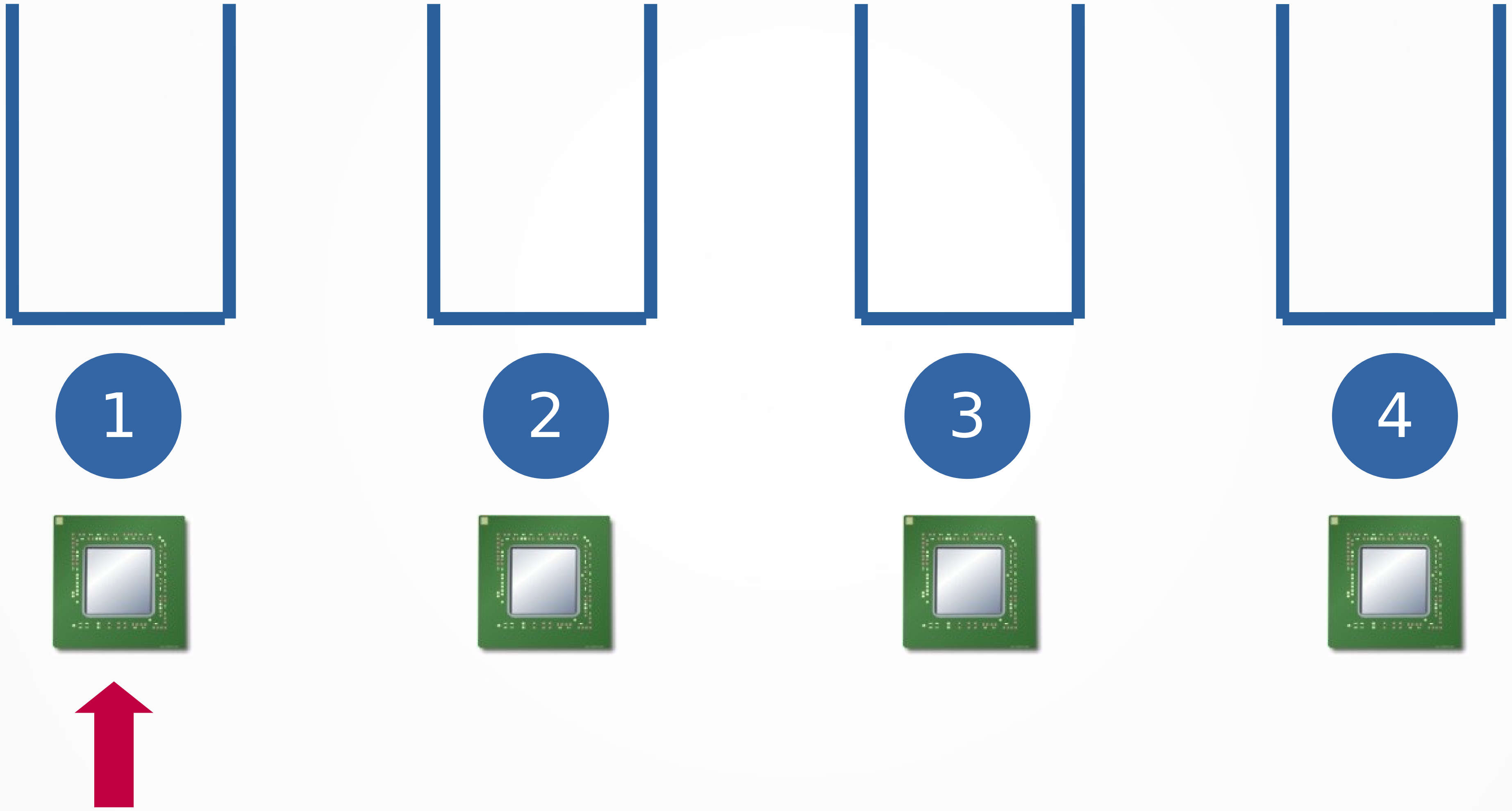
Second Model: Cilk

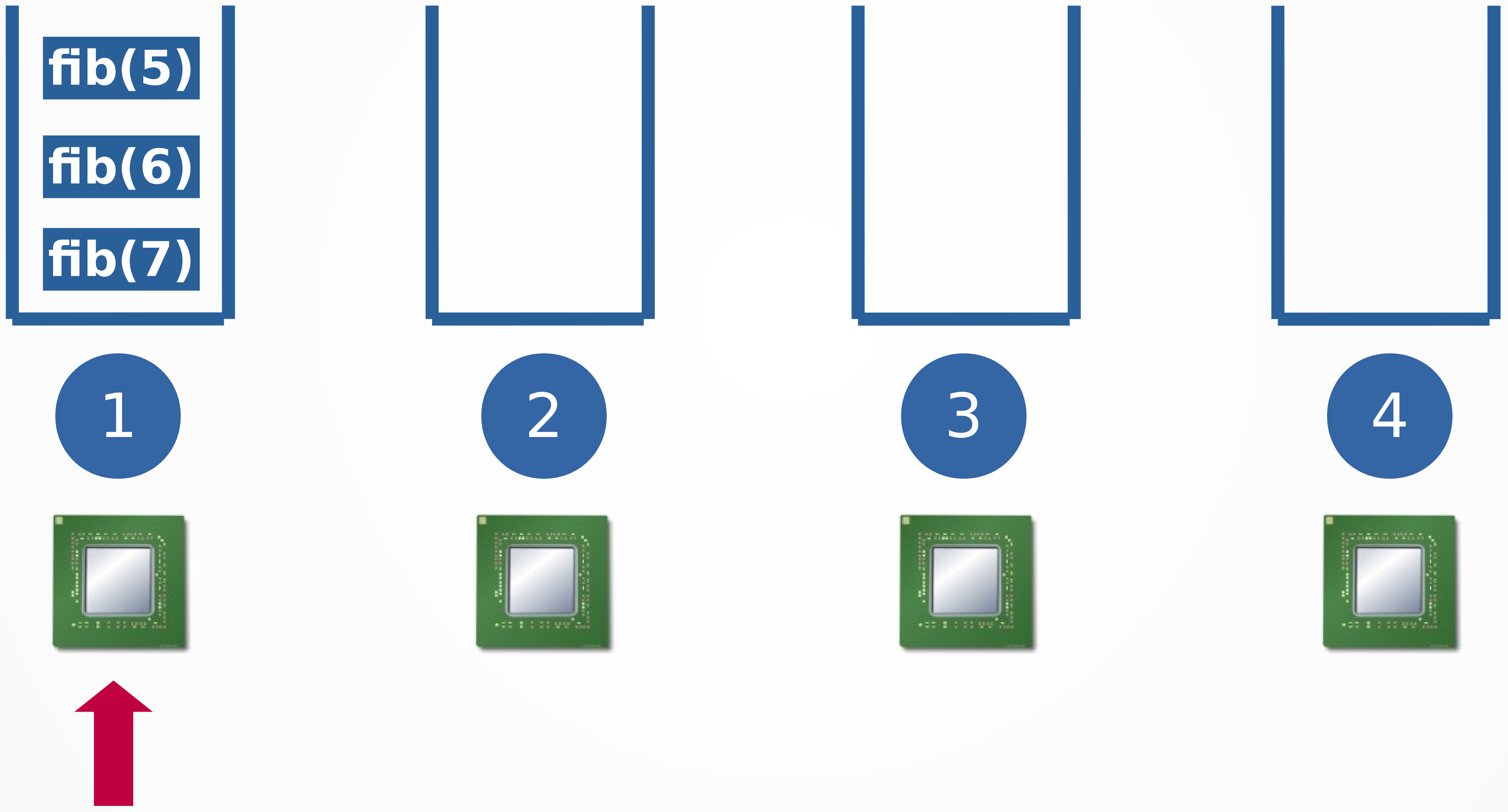
- Work through example (fib.c)

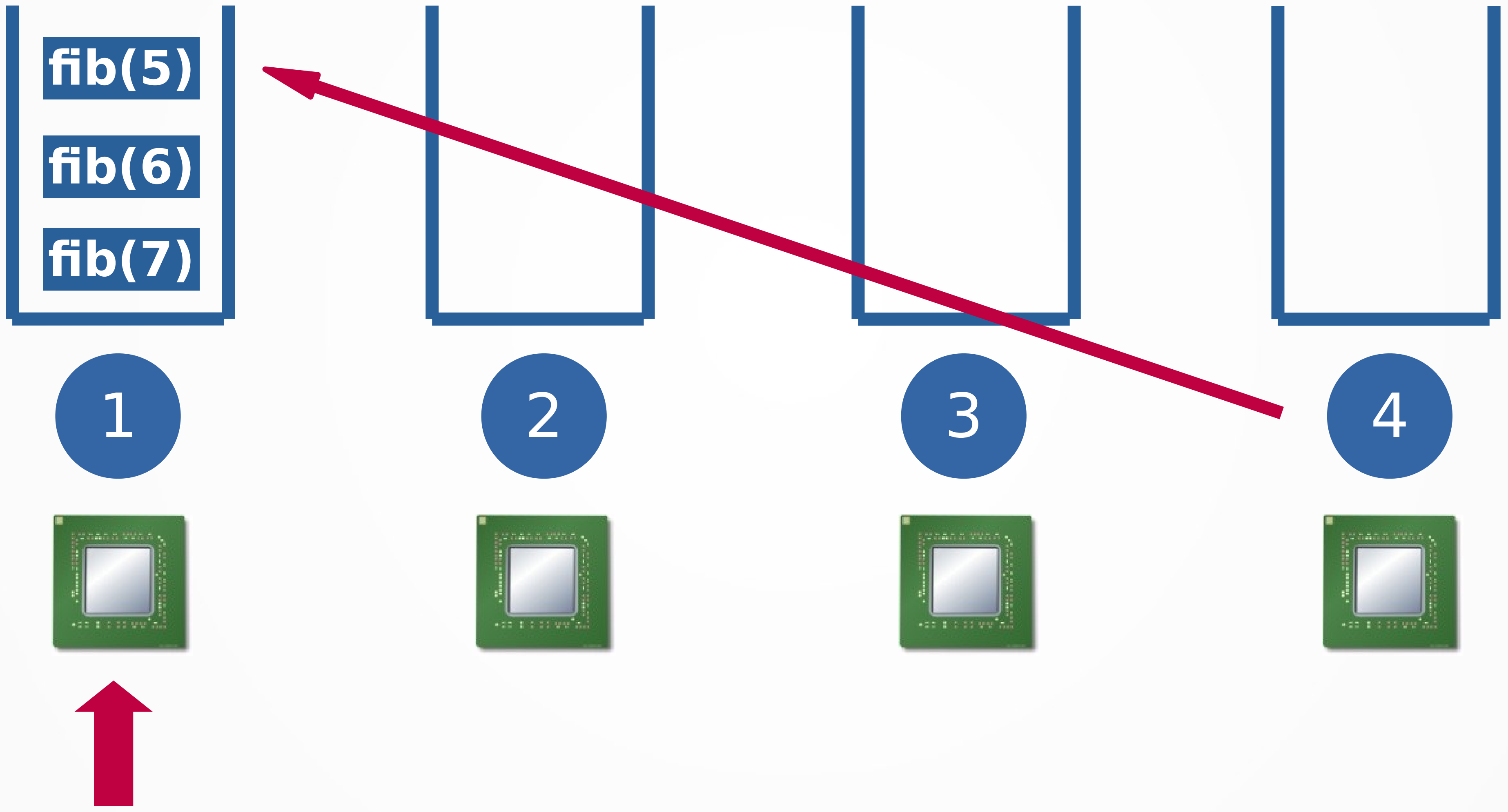
Second Model: Cilk

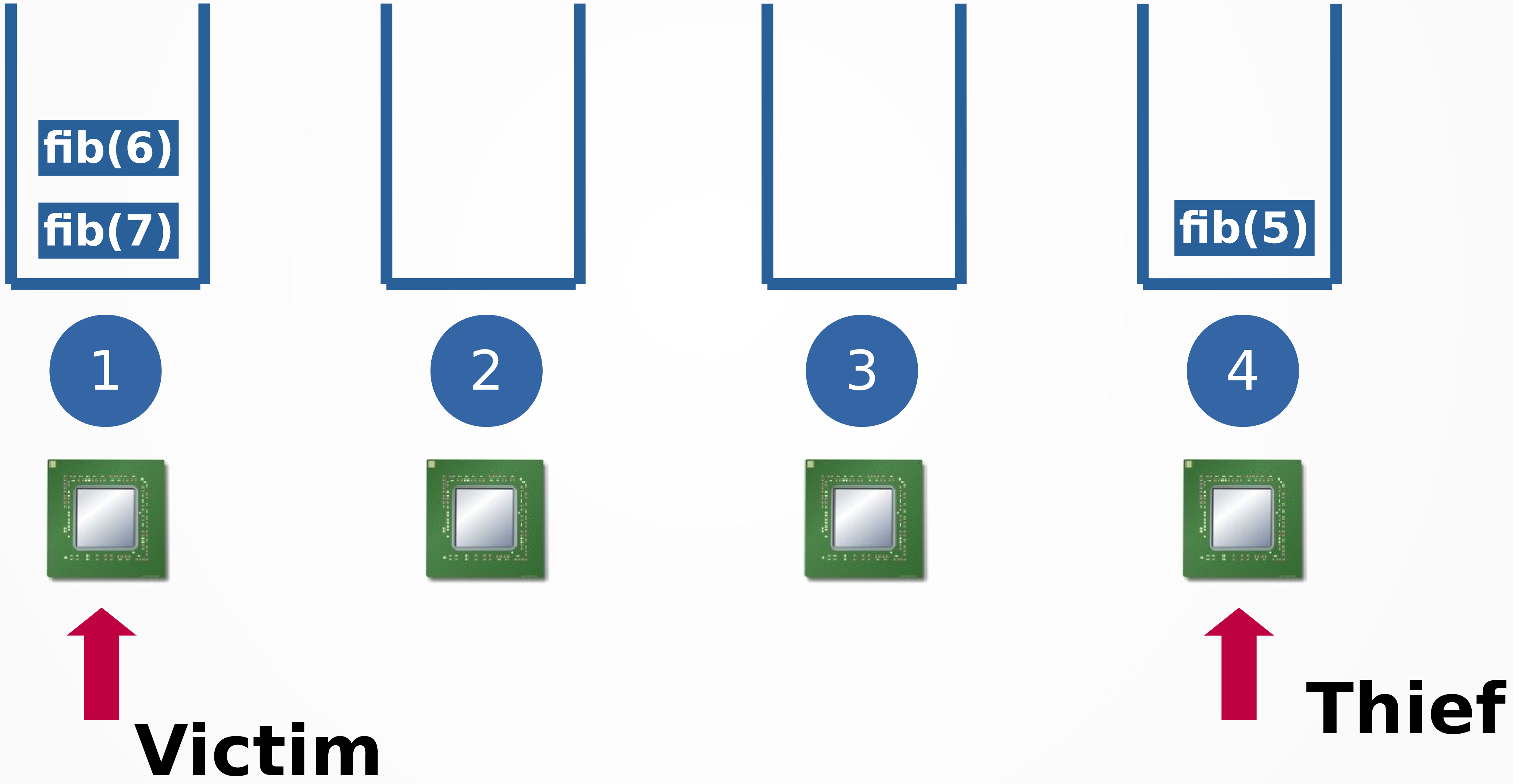
- Key to Cilk is the idea of *work stealing*…


1
2
3
4

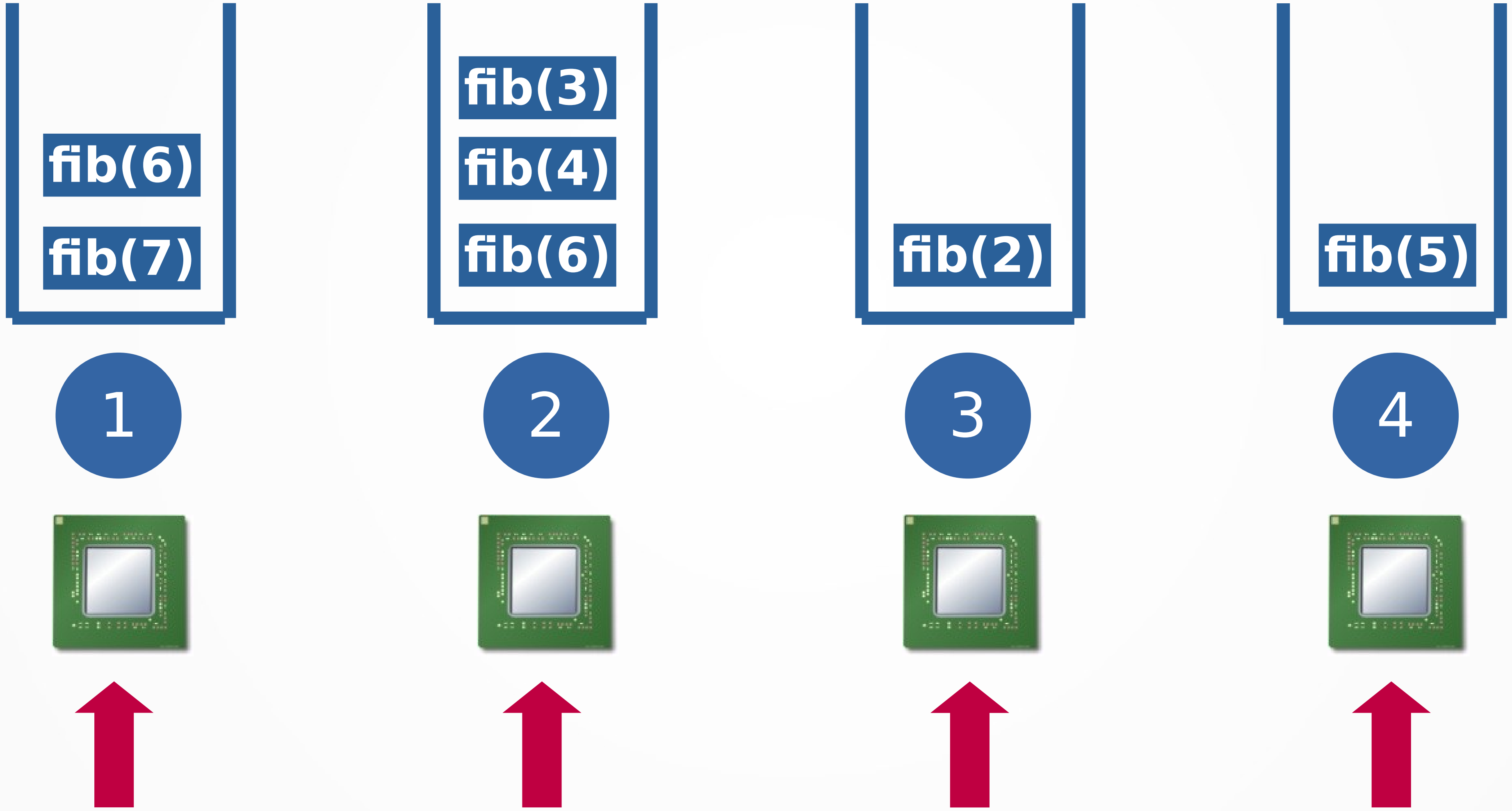

1
2
3
4


fib(5)
fib(6)
fib(7)
1
2
3
4

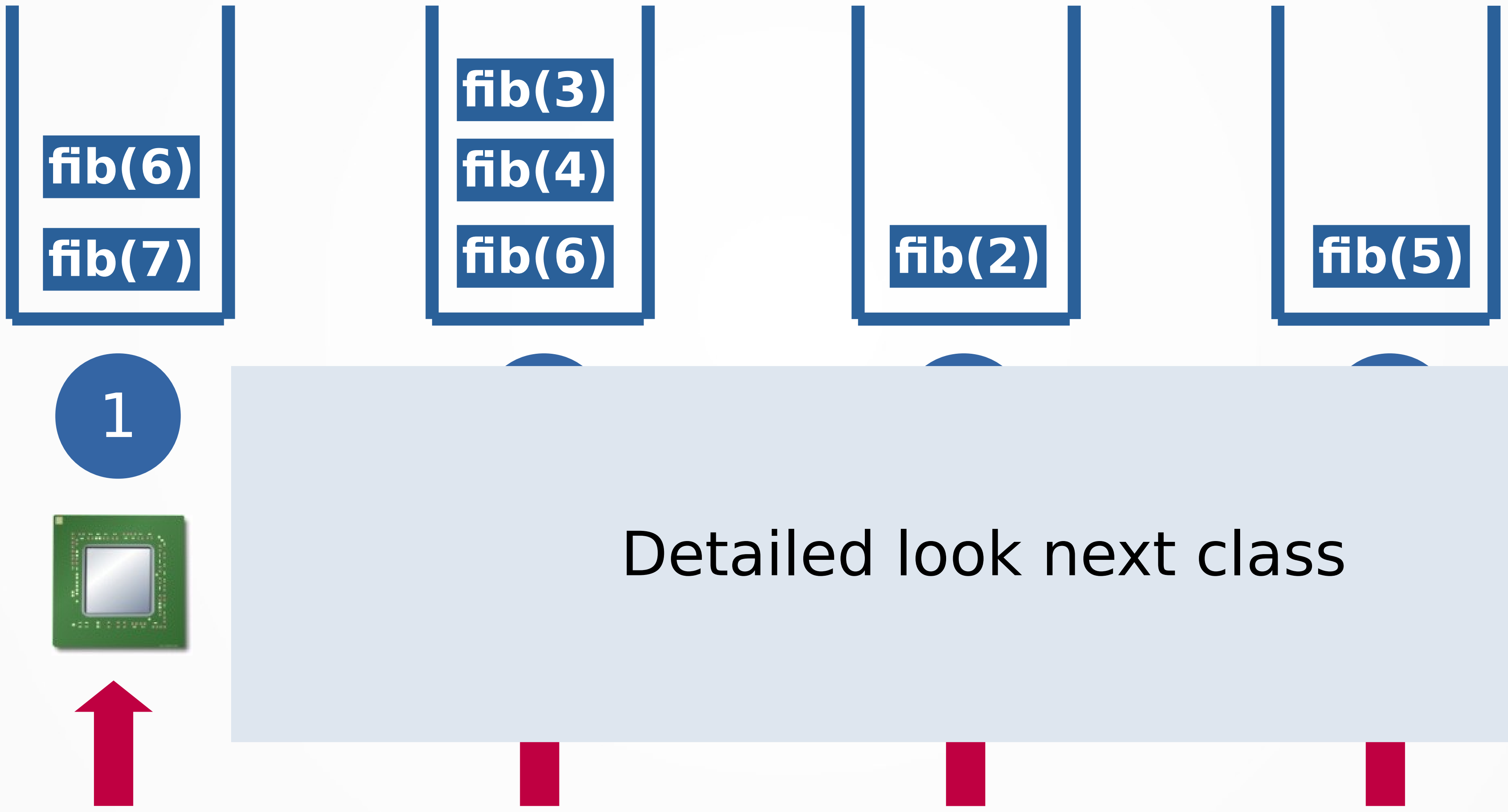

fib(5)
fib(6)
fib(7)
1
2
3
4


fib(6)
fib(7)
fib(5)
1
2
3
4
Victim
Thief

Dynamic Load Balancing



Dynamic Load Balancing



Detailed look next class

Second Model: Cilk

- Work through example (vector.c, merge.c)

For Reference / Self Study

- cilk statements
  - cilk_spawn
  - cilk_sync
  - cilk_for
- Compiling and linking
  - Use cilk header: #include <cilk/cilk.h>
  - Compile: gcc –std=c11 -fcilkplus code.c