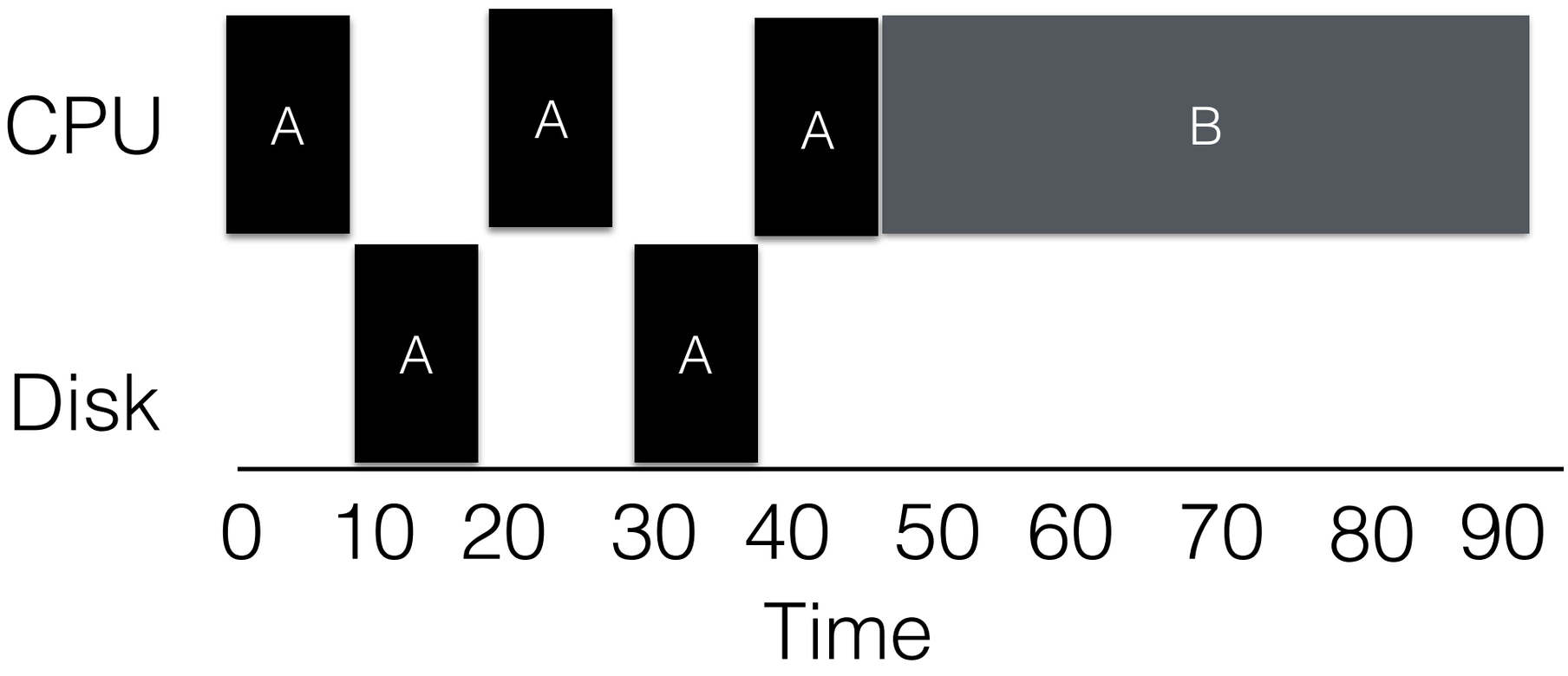
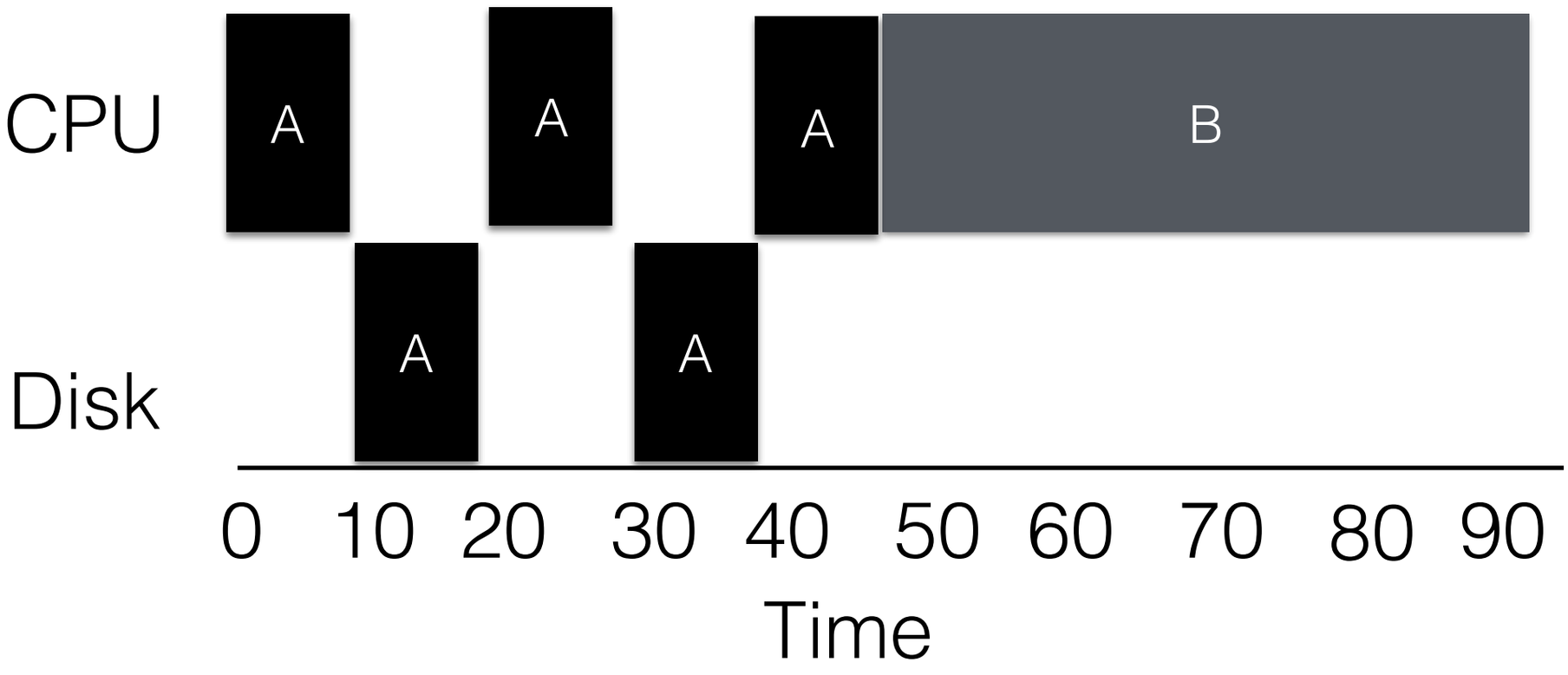
Operating Systems
Lecture 17: Concurrency

Nipun Batra
Sep 18, 2018

Increasing CPU Utilization

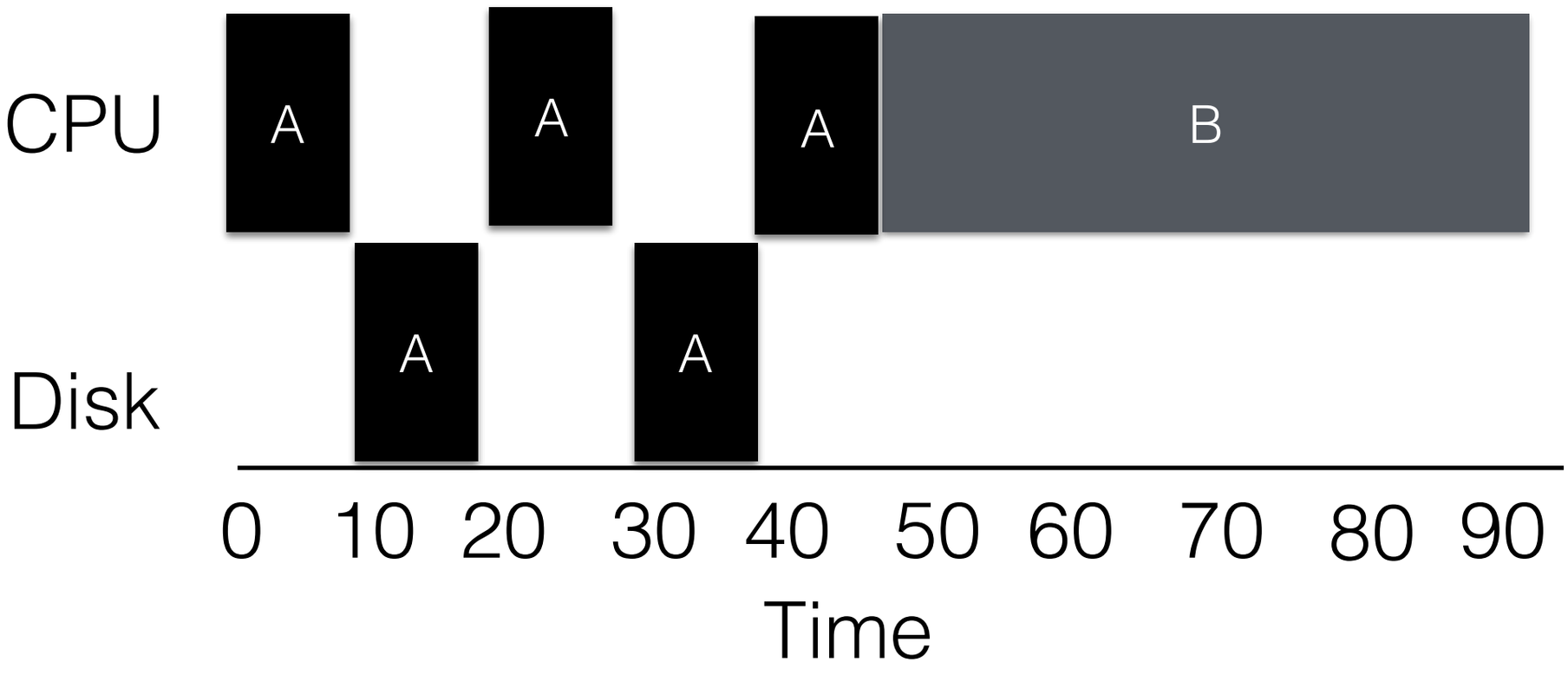


Increasing CPU Utilization



CPU utilisation (%) = (30+40)*100%/90 ~77%

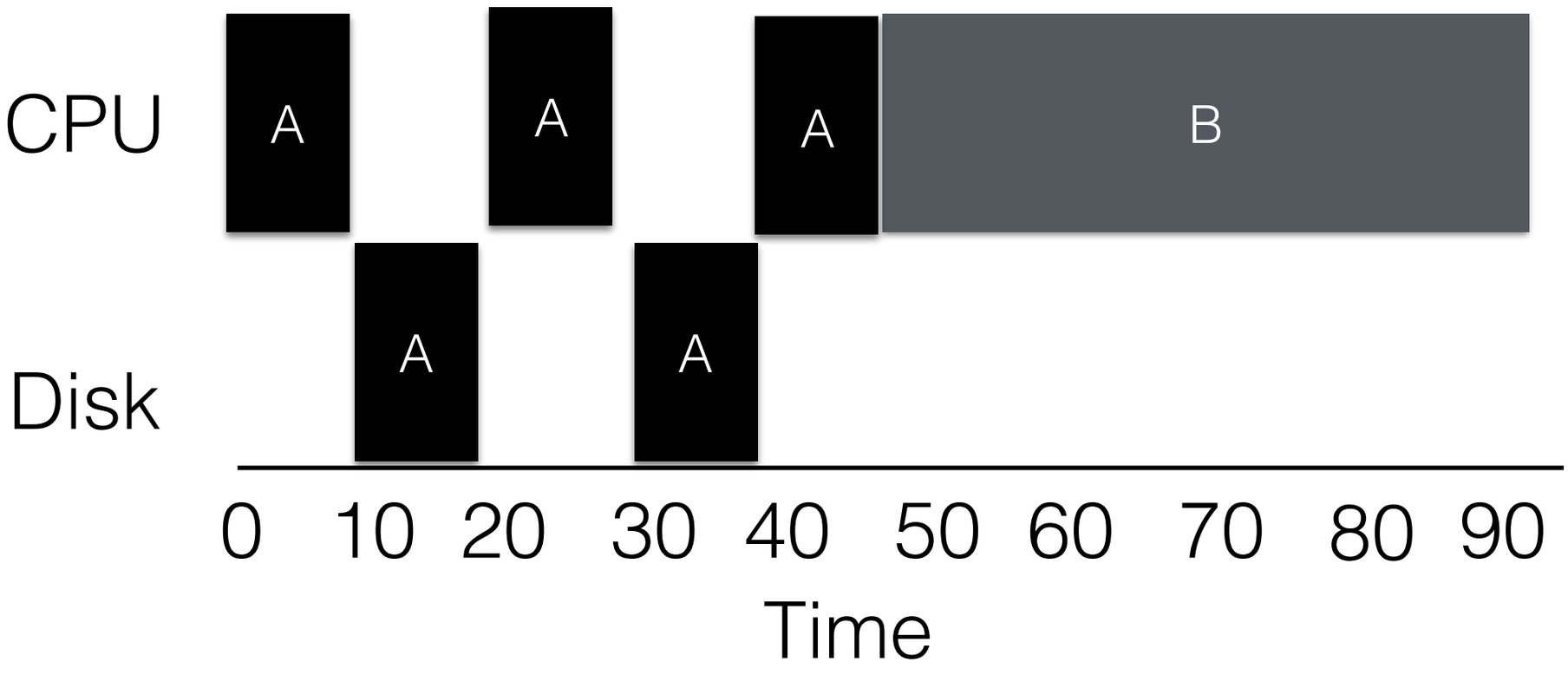
Increasing CPU Utilization



CPU utilisation (%) = (30+40)*100%/90 ~77%

Avg. Response Time = (0+50)/2 =25

Increasing CPU Utilization

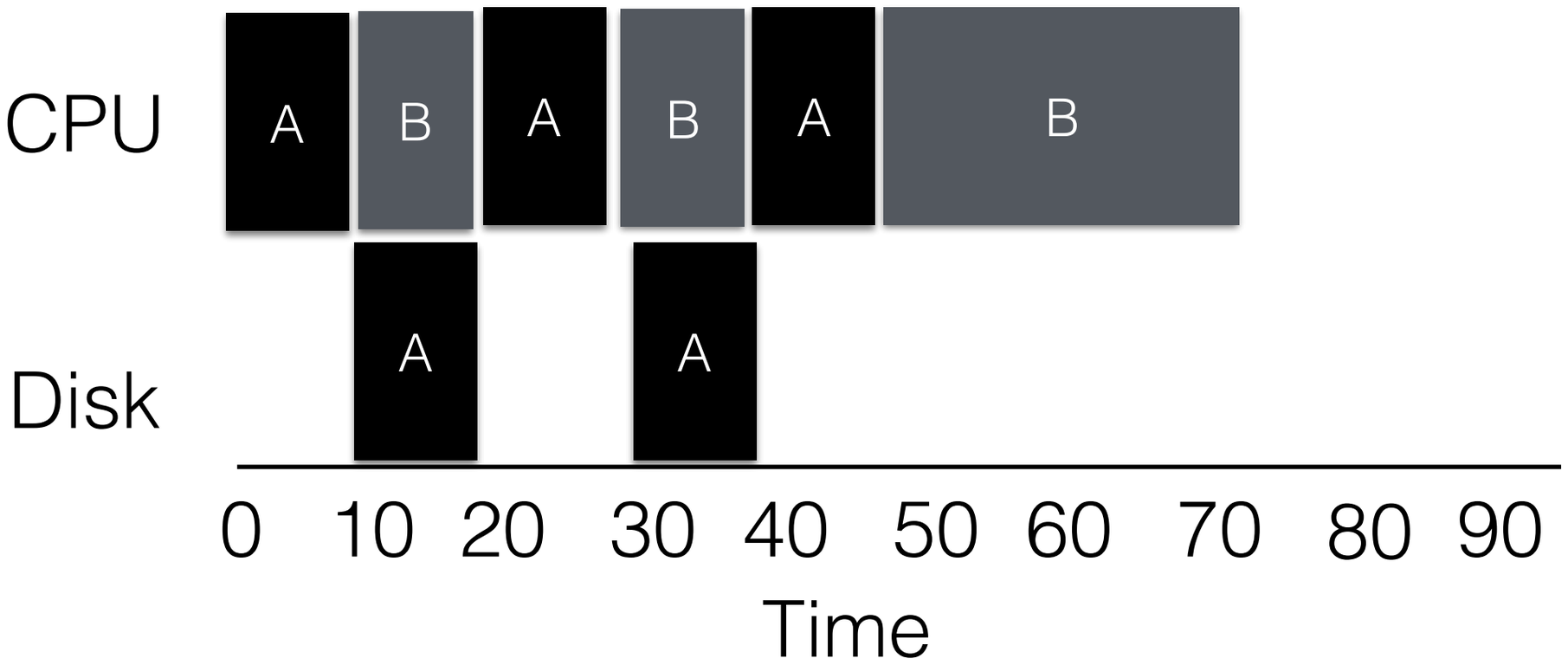


CPU utilisation (%) = (30+40)*100%/90 ~77%

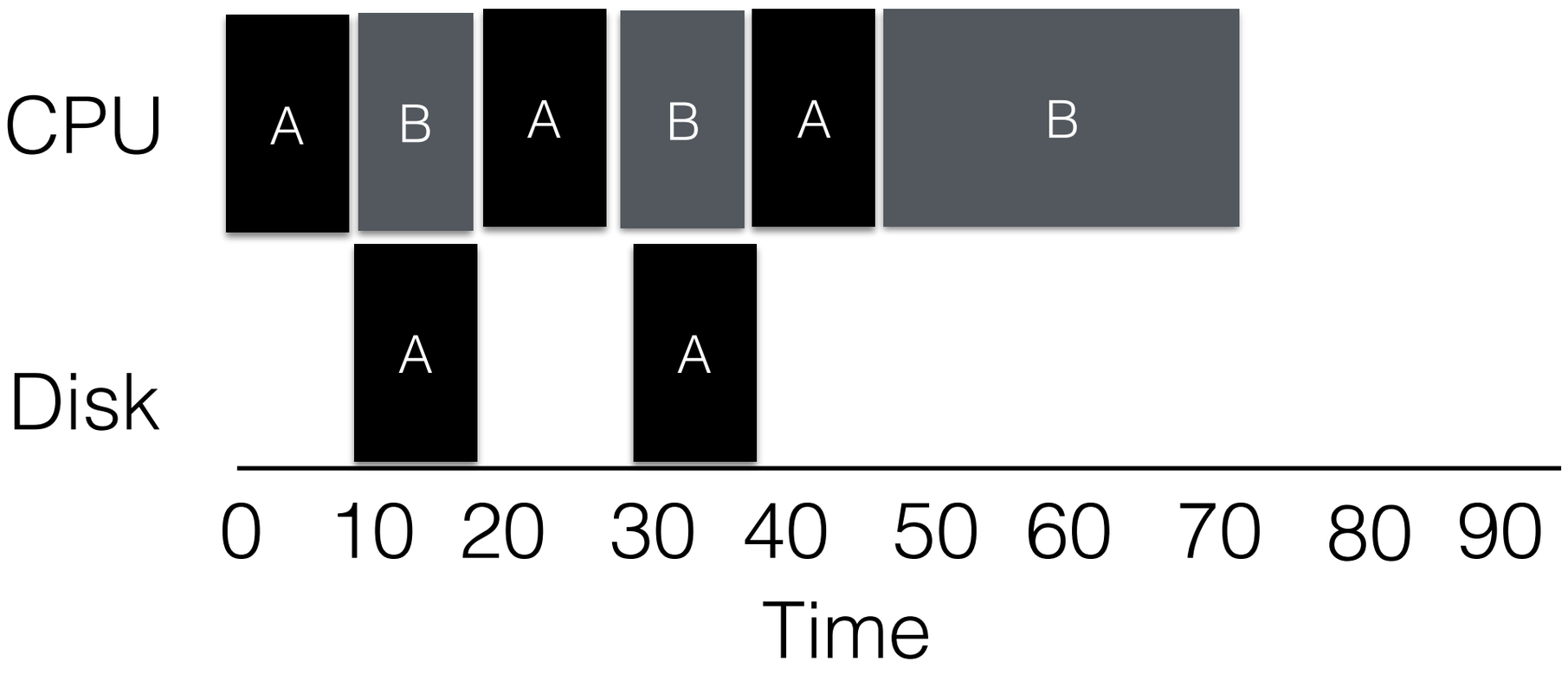
Avg. Response Time = (0+50)/2 =25

Avg. Turnaround Time = (50+90)/2 =70

Increasing CPU Utilization

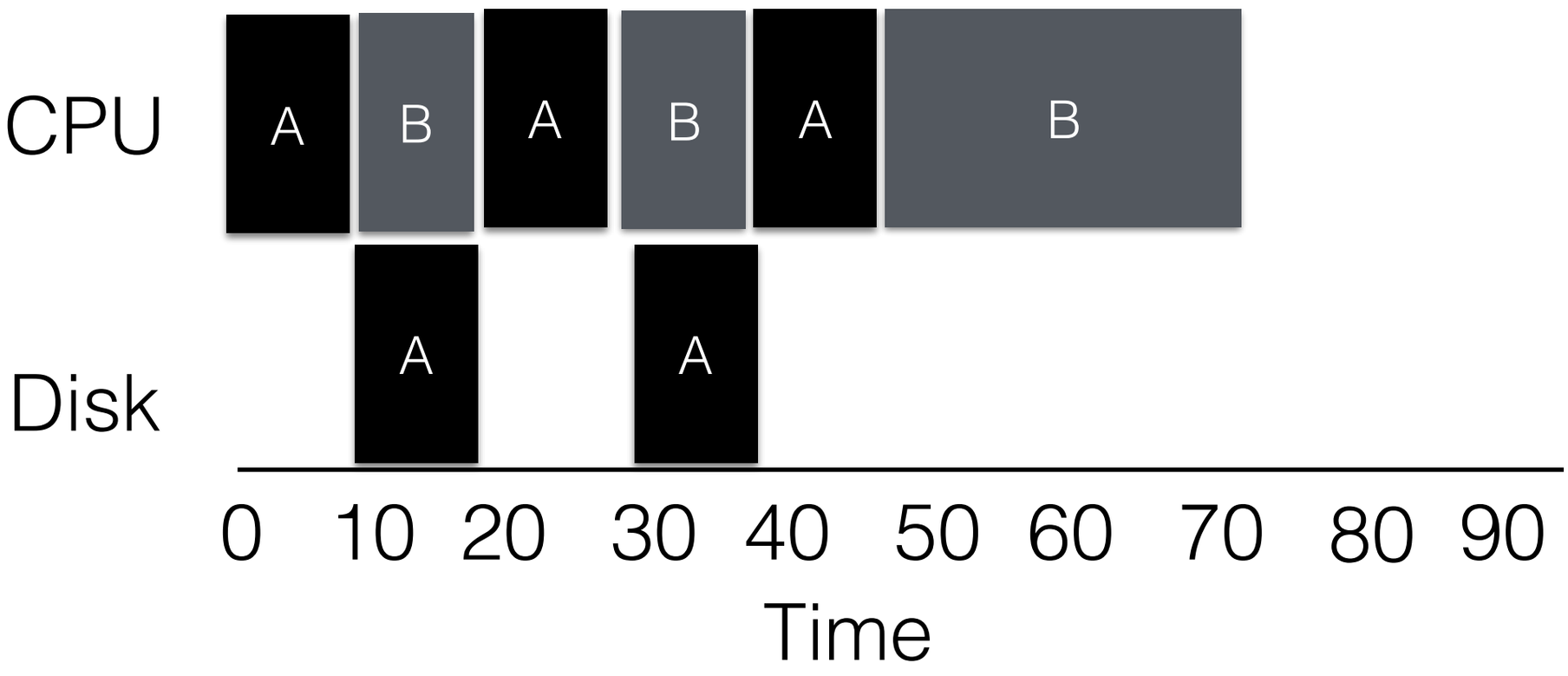


Increasing CPU Utilization



CPU utilisation (%) = (30+40)*100%/70 =100%

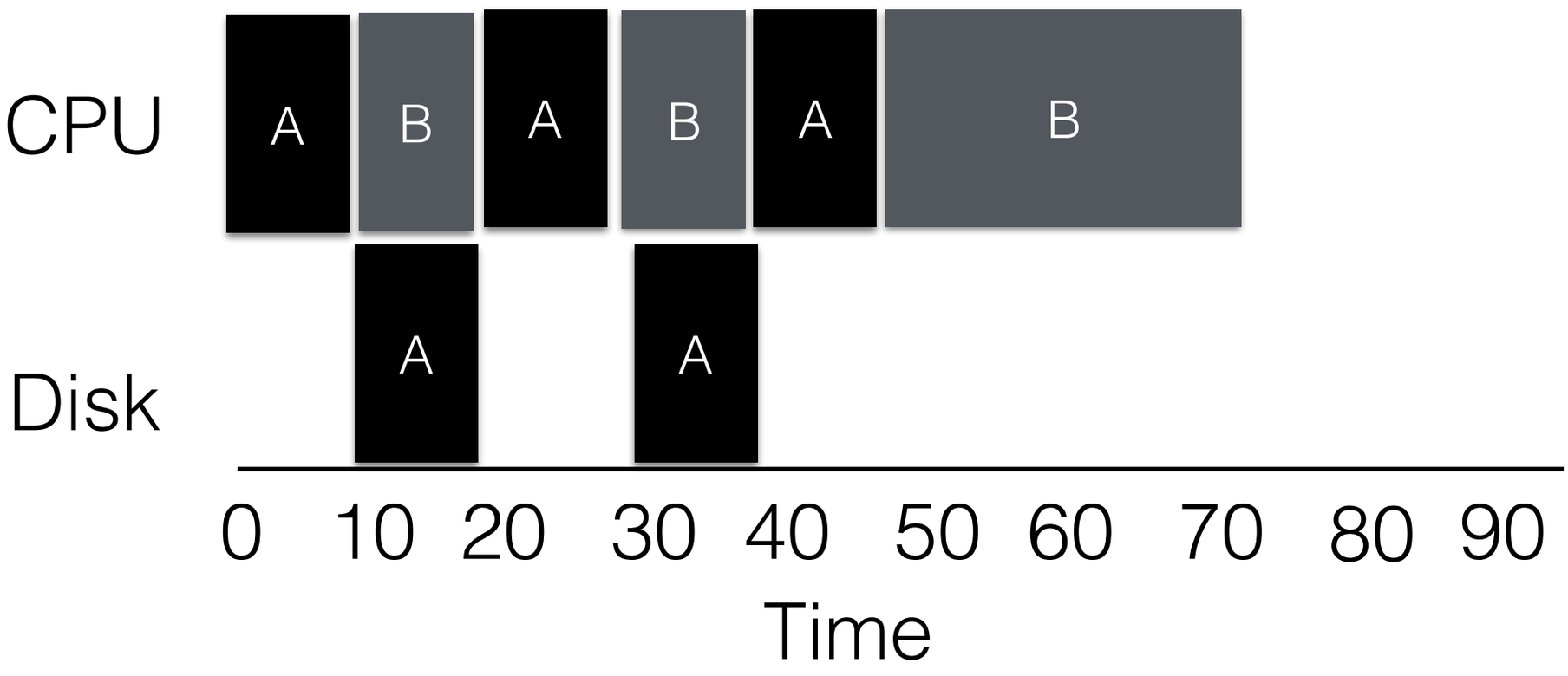
Increasing CPU Utilization



CPU utilisation (%) = (30+40)*100%/70 =100%

Avg. Response Time = (0+10)/2 =5

Increasing CPU Utilization

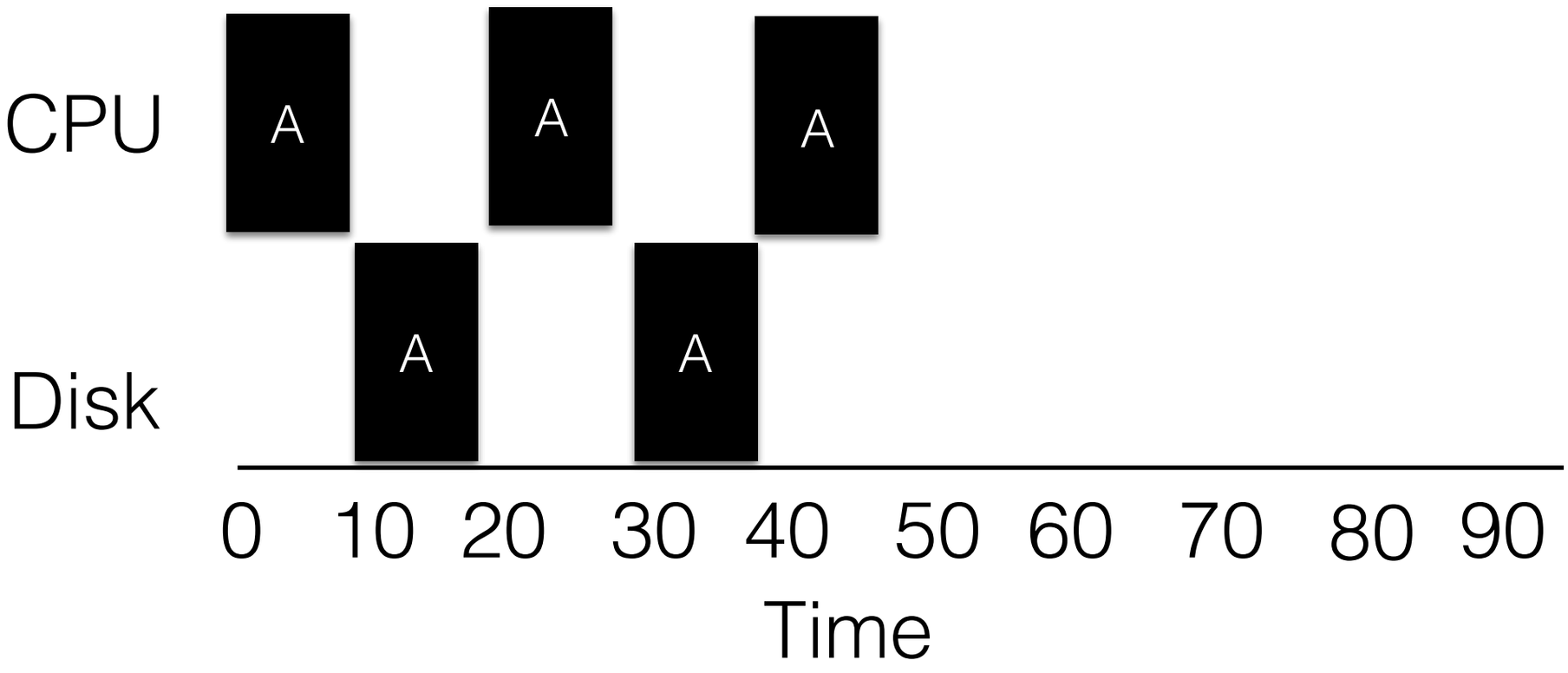


CPU utilisation (%) = (30+40)*100%/70 =100%

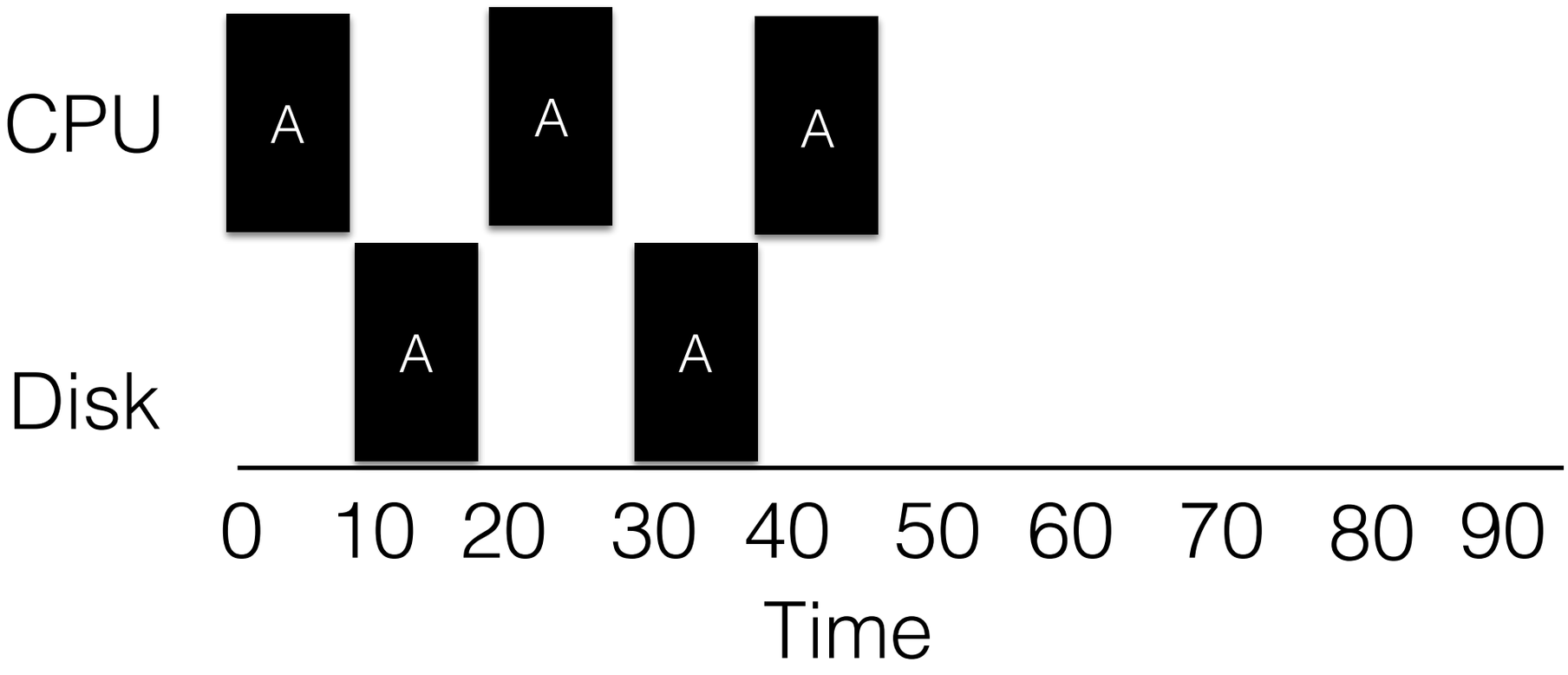
Avg. Response Time = (0+10)/2 =5

Avg. Turnaround Time = (50+70)/2 =60

What if there's a single process?

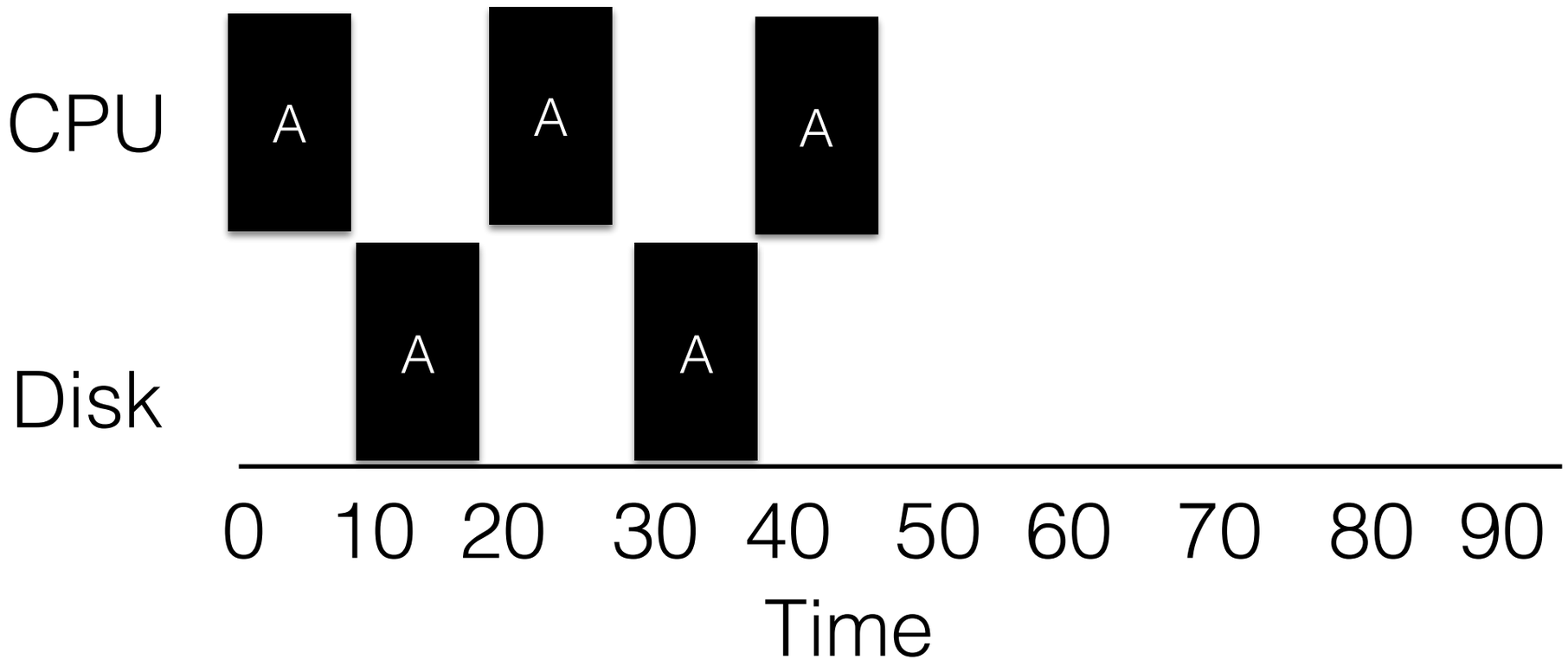


What if there's a single process?

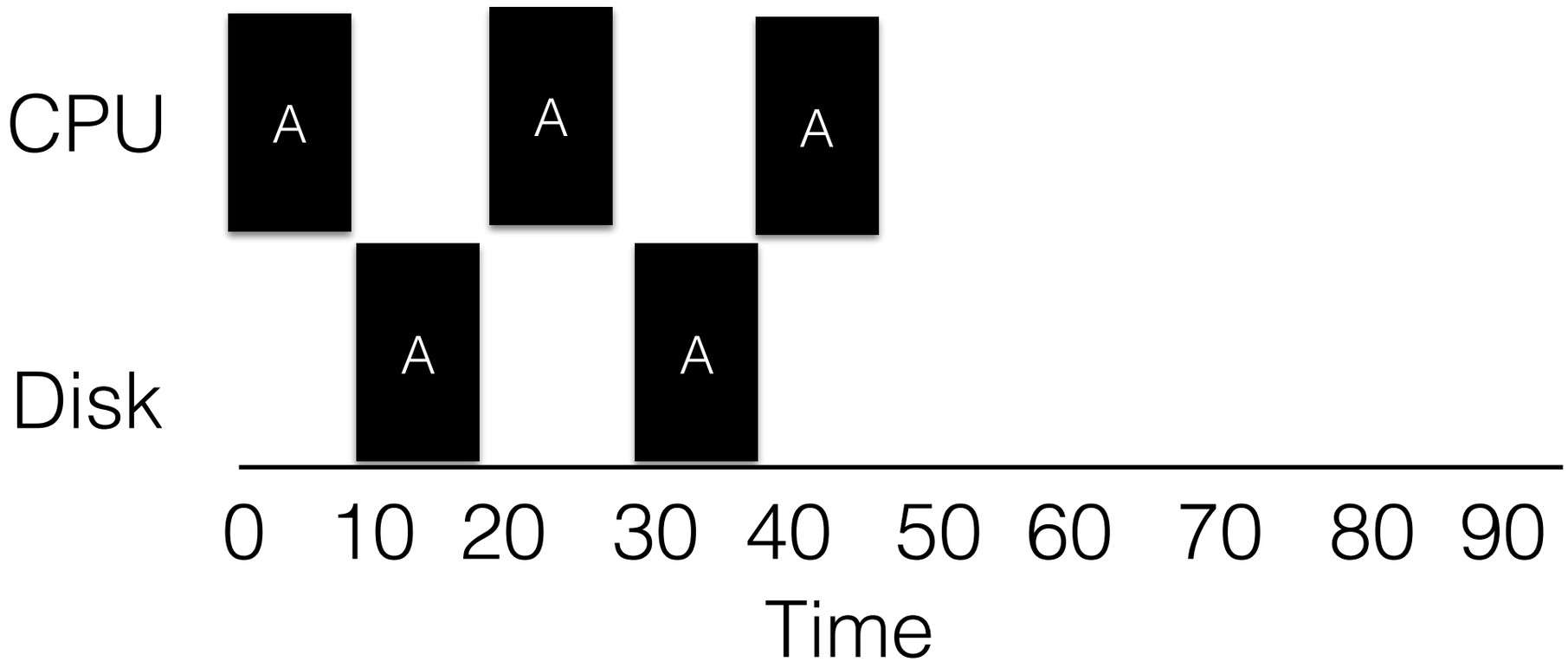


CPU utilisation (%) = (30)*100%/50 ~60%

How can we reduce turnaround?

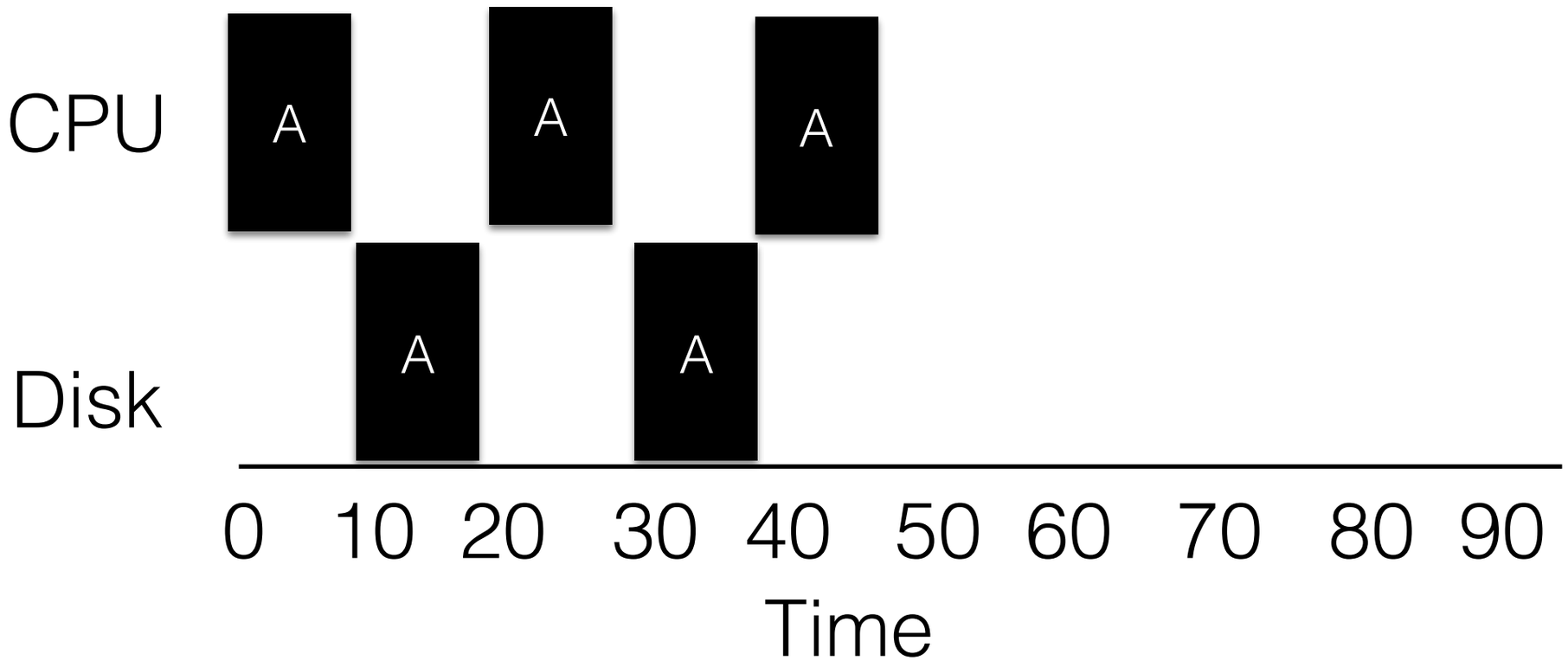


How can we reduce turnaround?

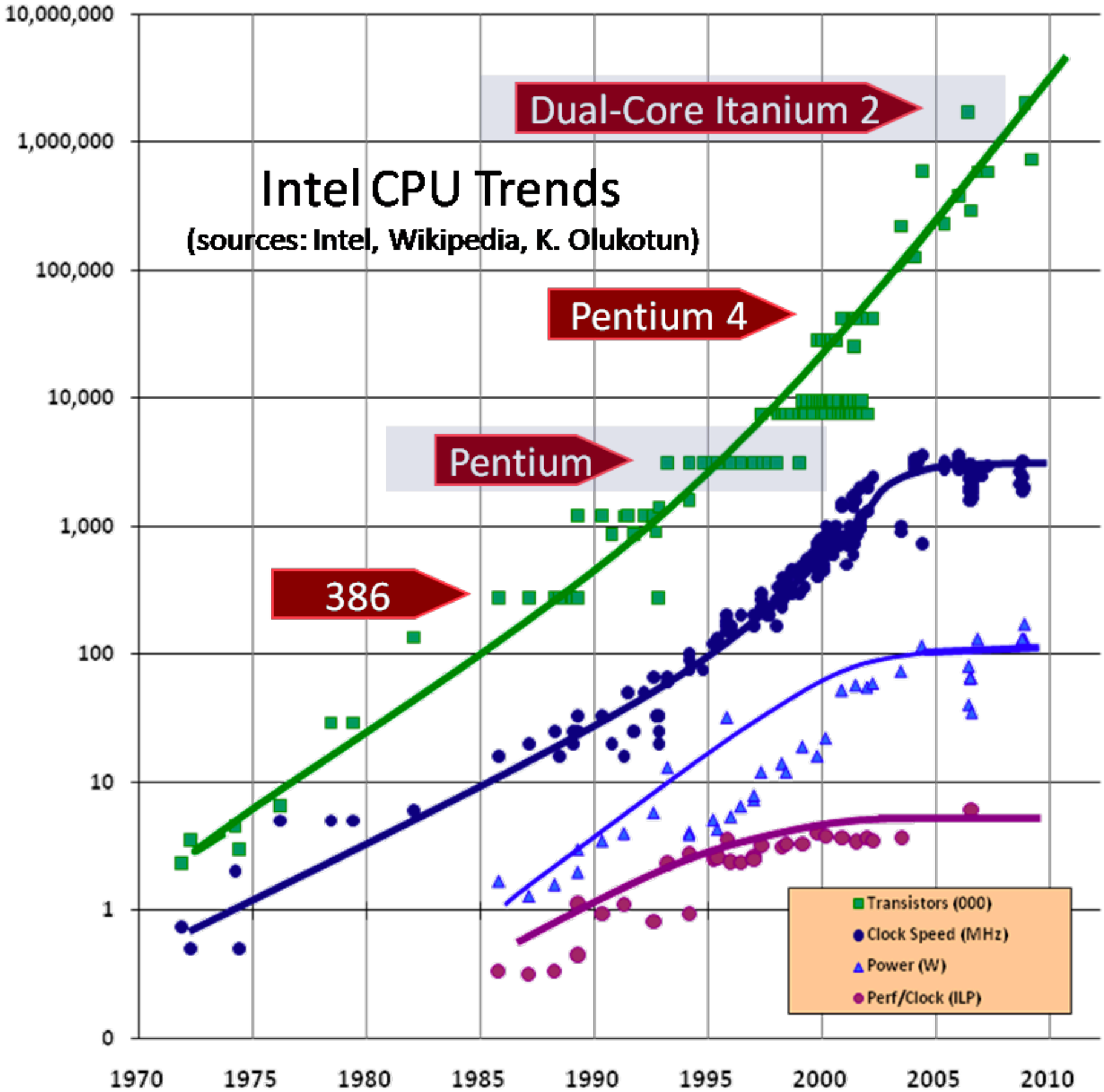


1. Increase the clock speed?

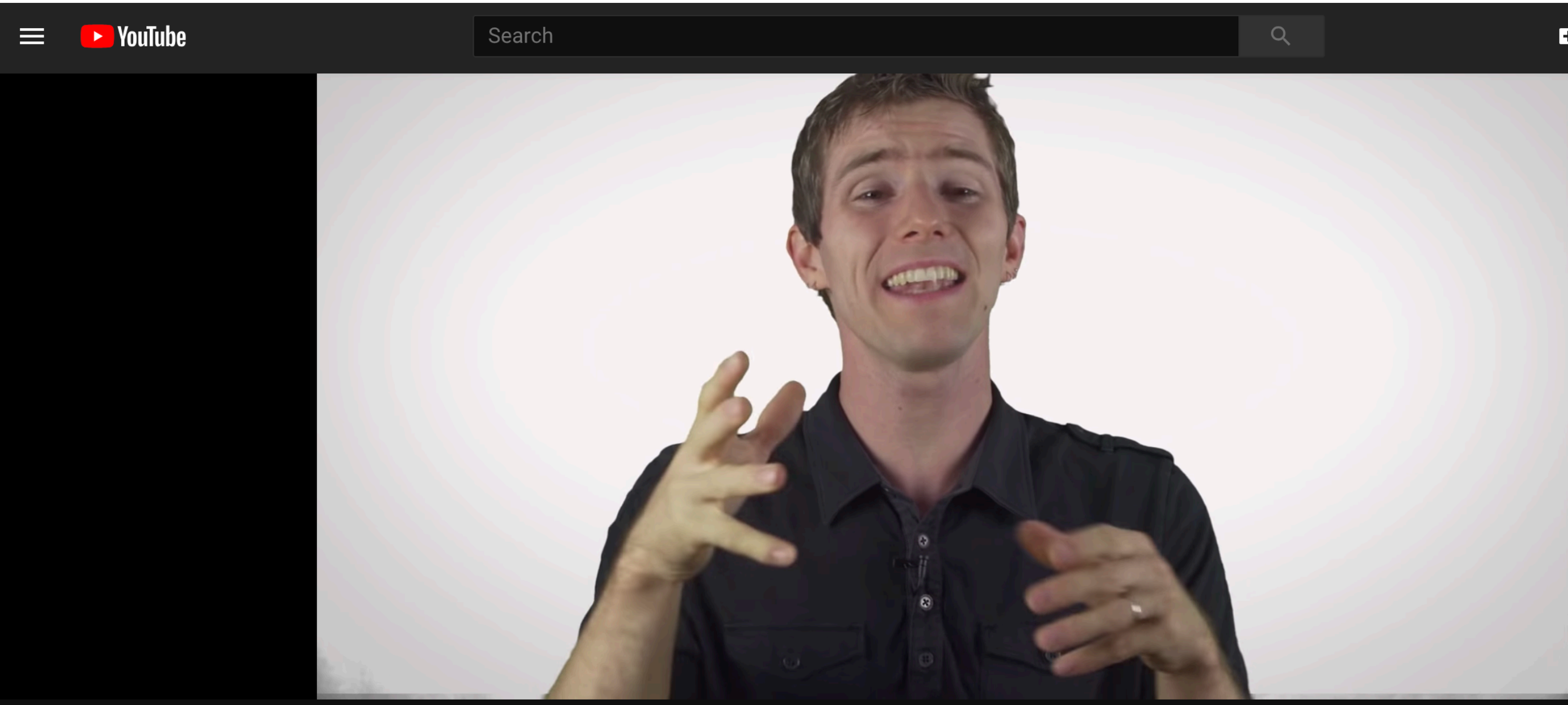
How can we reduce turnaround?



1. Increase the clock speed?
2. Increase work/clock cycle.


Intel CPU Trends
(sources: Intel, Wikipedia, K. Olukotun)
Dual-Core Itanium 2
Pentium 4
Pentium
386
Transistors (000)
Clock Speed (MHz)
Power (W)
Perf/Clock (ILP)
10,000,000
1,000,000
100,000
10,000
1,000
100
10
1
0
1970
1975
1980
1985
1990
1995
2000
2005
2010

Hyper threading



Hyper-threading

present, the operating system addresses two virtual (logical) cores and shares the workload between them when possible. The main function of hyper-threading is to increase the number of independent instructions in the pipeline; it takes advantage of superscalar architecture, in which multiple instructions operate on separate data in parallel. With HTT, one physical core appears as two processors to the operating system, allowing concurrent scheduling of two processes per core. In addition, two or more processes can use the same resources: if resou available, then another process can continue if its resources are avail

In this high-level depictic fetched from RAM (differ instructions of four differ reordered by the front er bubbles), and passed to executing instructions fr the same clock cycle.[1][2

In addition to requiring simultaneous multithreading (SMT) support in t

Hyper-threading

Drawbacks [edit]

When the first HT processors were released, many operating systems were not optimized for hyper-threading technology (e.g. Windows 2000 and Linux older than 2.4).[24]

In 2006, hyper-threading was criticised for energy inefficiency.[25] For example, specialist low-power CPU design company ARM stated that simultaneous multithreading (SMT) can use up to 46% more power than ordinary dual-core designs. Furthermore, they claimed that SMT increases cache thrashing by 42%, whereas dual core results in a 37% decrease.[26] Intel disputed this claim, stating that hyper-threading is highly efficient because it uses resources that would otherwise be idle or underutilised.[*citation needed*]

In 2010, ARM said it might include simultaneous multithreading in its future chips;[27] however, this was rejected in favor of their 2012 64-bit design.[28]

In 2013, Intel dropped SMT in favor of out-of-order execution for its Silvermont processor cores, as they found this gave better performance with better power efficiency than a lower number of cores with SMT.[29]

In 2017, it was revealed Intel's Skylake and Kaby Lake processors had a bug with their implementation of hyper-threading that could cause data loss.[30] Microcode updates were later released to address the issue.[31]

Hyper-threading

CACHE MISSING FOR FUN AND PROFIT

COLIN PERCIVAL

ABSTRACT. Simultaneous multithreading — put simply, the sharing of the execution resources of a superscalar processor between multiple execution threads — has recently become widespread via its introduction (under the name "Hyper-Threading") into Intel Pentium 4 processors. In this implementation, for reasons of efficiency and economy of processor area, the sharing of processor resources between threads extends beyond the execution units; of particular concern is that the threads share access to the memory caches.

We demonstrate that this shared access to memory caches provides not only an easily used high bandwidth covert channel between threads, but also permits a malicious thread (operating, in theory, with limited privileges) to monitor the execution of another thread, allowing in many cases for theft of cryptographic keys.

Finally, we provide some suggestions to processor designers, operating system vendors, and the authors of cryptographic software, of how this attack could be mitigated or eliminated entirely.

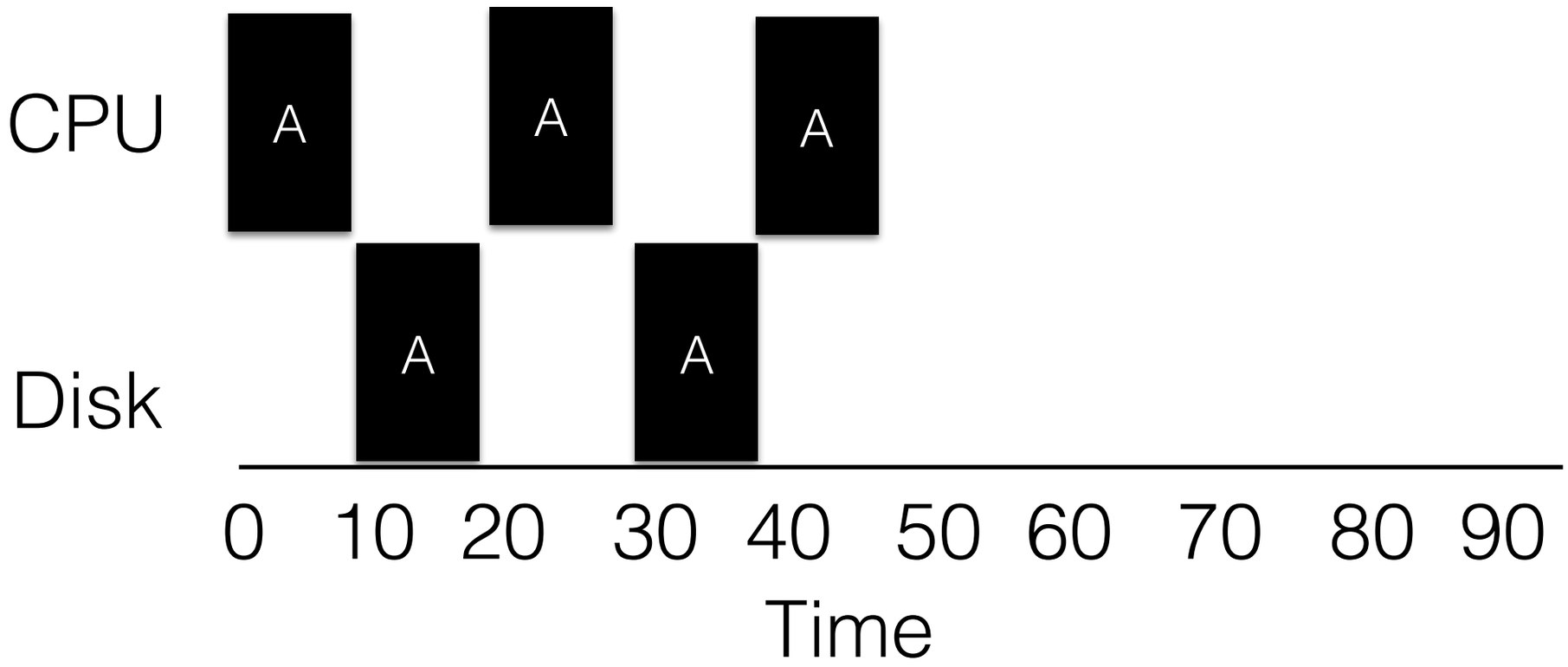
Multi-core


YouTube
Search
CPU '11 2930-04
STL82CK MALAY
3.80GHZ/8M/6.40
453018295 S2
0:07 / 5:59
How Do CPUs Use Multiple Cores?
1,043,412 views
22K
303
SHARE
Up next

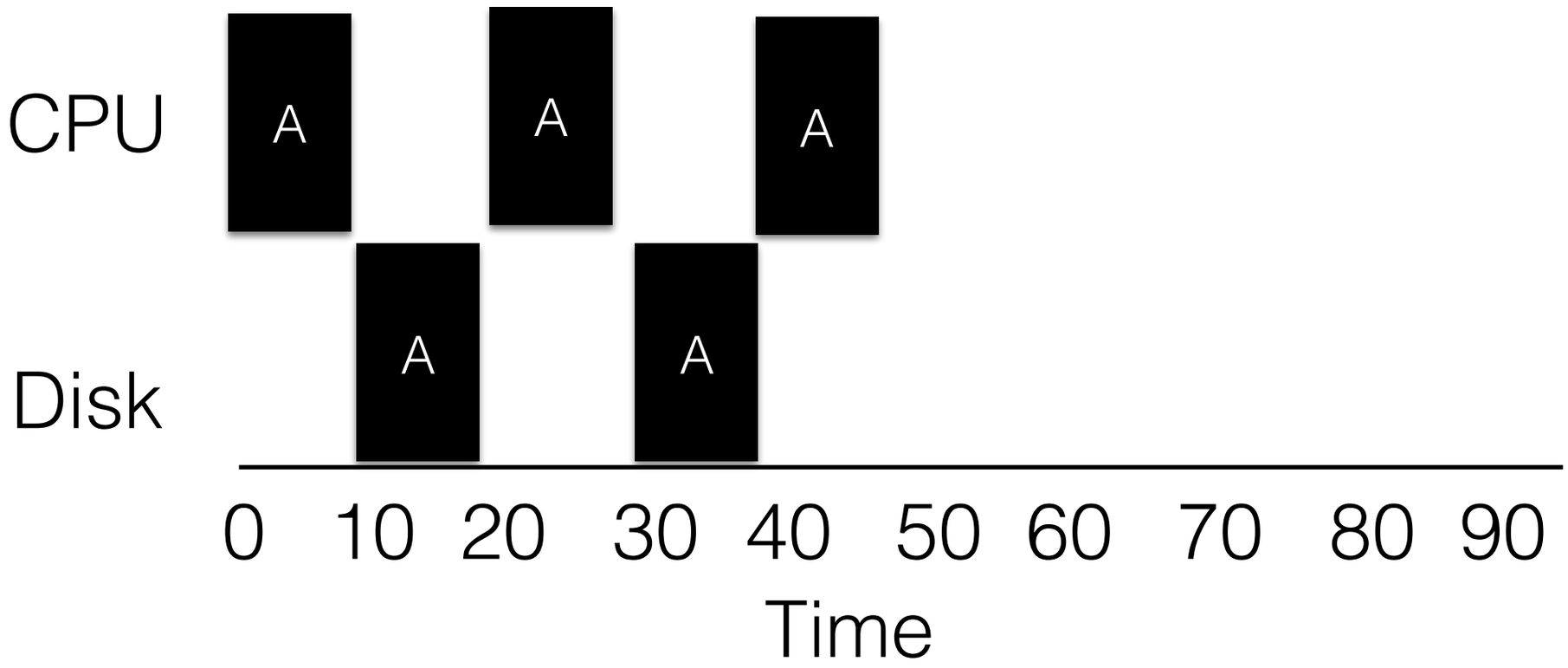
More reading …

1. https://www.howtogeek.com/194756/cpu-basics-multiple-cpus-cores-and-hyper-threading-explained/
2. https://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html
3. https://ark.intel.com/Search/FeatureFilter?productType=processors
4. http://www.gotw.ca/publications/concurrency-ddj.htm
5. https://smoothspan.com/2007/09/06/a-picture-of-the-multicore-crisis/

How can we reduce turnaround?

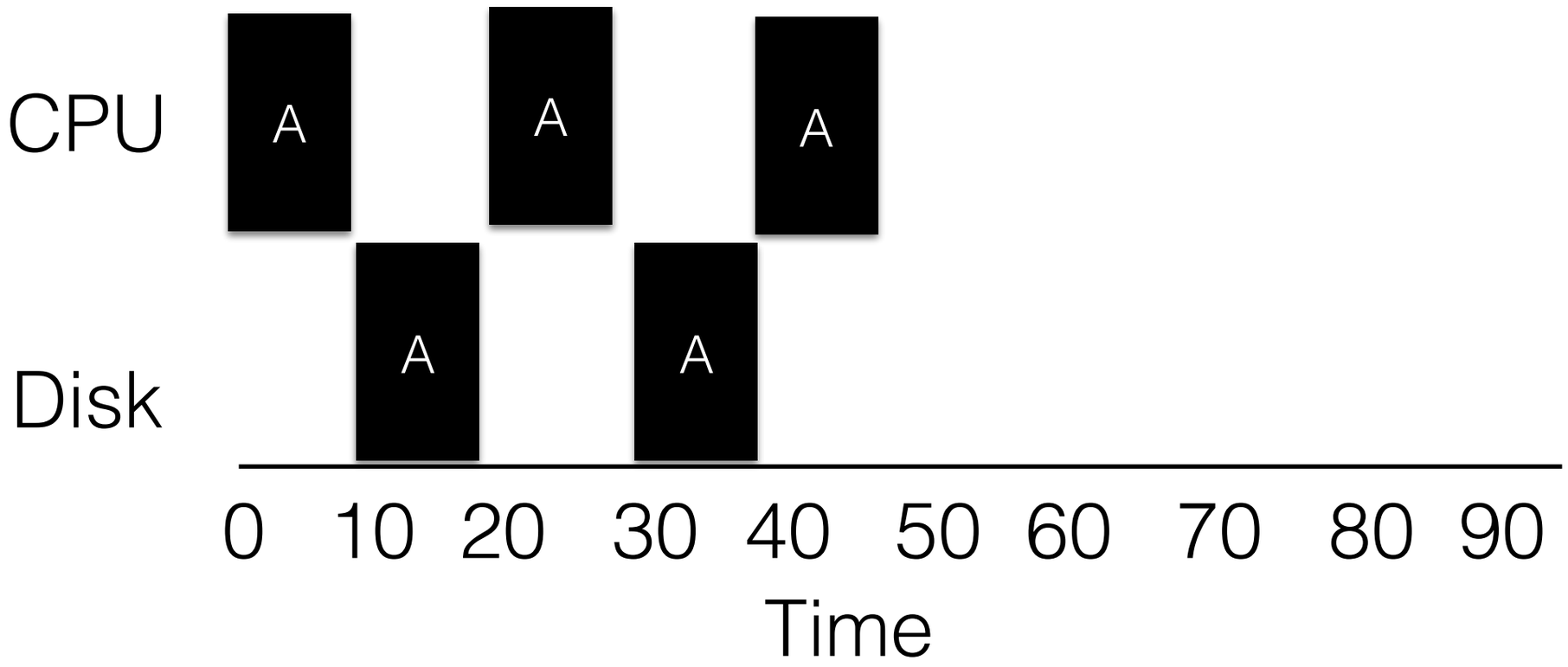


How can we reduce turnaround?



1. Concurrent execution

How can we reduce turnaround?



1. Concurrent execution
    1. Write applications that can fully utilize CPU(s)

Strategy 1

Strategy 1

1. Build applications from many communicating processes

Strategy 1

1. Build applications from many communicating processes
    1. Eg - Google Chrome?!

Strategy 1

1. Build applications from many communicating processes
    1. Eg - Google Chrome?!
2. Communicate using IPC:

Strategy 1

1. Build applications from many communicating processes
    1. Eg - Google Chrome?!
2. Communicate using IPC:
    1. files?!

Strategy 1

1. Build applications from many communicating processes
    1. Eg - Google Chrome?!
2. Communicate using IPC:
    1. files?!
    2. Pipes

Strategy 1

1. Build applications from many communicating processes
    1. Eg - Google Chrome?!
2. Communicate using IPC:
    1. files?!
    2. Pipes
3. ...

Strategy 1

1. Build applications from many communicating processes
    1. Eg - Google Chrome?!
2. Communicate using IPC:
    1. files?!
    2. Pipes
    3. …
3. Multiple copies of address space!

Sidenote - Google Chrome Process Models

Sidenote - Google Chrome Process Models

1. Process per site instance:

Sidenote - Google Chrome Process Models

1. Process per site instance:
    1. Site - domain (google.com, mail.google.com, docs.google.com. all same site)

Sidenote - Google Chrome Process Models

1. Process per site instance:
    1. Site - domain (google.com, mail.google.com, docs.google.com. all same site)
2. Process per site

Sidenote - Google Chrome Process Models

1. Process per site instance:
    1. Site - domain (google.com, mail.google.com, docs.google.com. all same site)
2. Process per site
3. Process per tab

Sidenote - Google Chrome Process Models

1. Process per site instance:
    1. Site - domain (google.com, mail.google.com, docs.google.com. all same site)
2. Process per site
3. Process per tab
4. Single process

Sidenote - Google Chrome Process Models

1. Process per site instance:
    1. Site - domain (google.com, mail.google.com, docs.google.com. all same site)
2. Process per site
3. Process per tab
4. Single process
5. More info here: https://www.chromium.org/developers/design-documents/process-models

Sidenote - Google Chrome Process Models

1. Process per site instance:
    1. Site - domain (google.com, mail.google.com, docs.google.com. all same site)
2. Process per site
3. Process per tab
4. Single process
5. More info here: https://www.chromium.org/developers/design-documents/process-models
6. Will be part of upcoming homework to illustrate different process models of Chrome/Chromium

Strategy 2

Strategy 2

1. Use Threads : just like processes, but, share the address space (i.e. PT)

Strategy 2

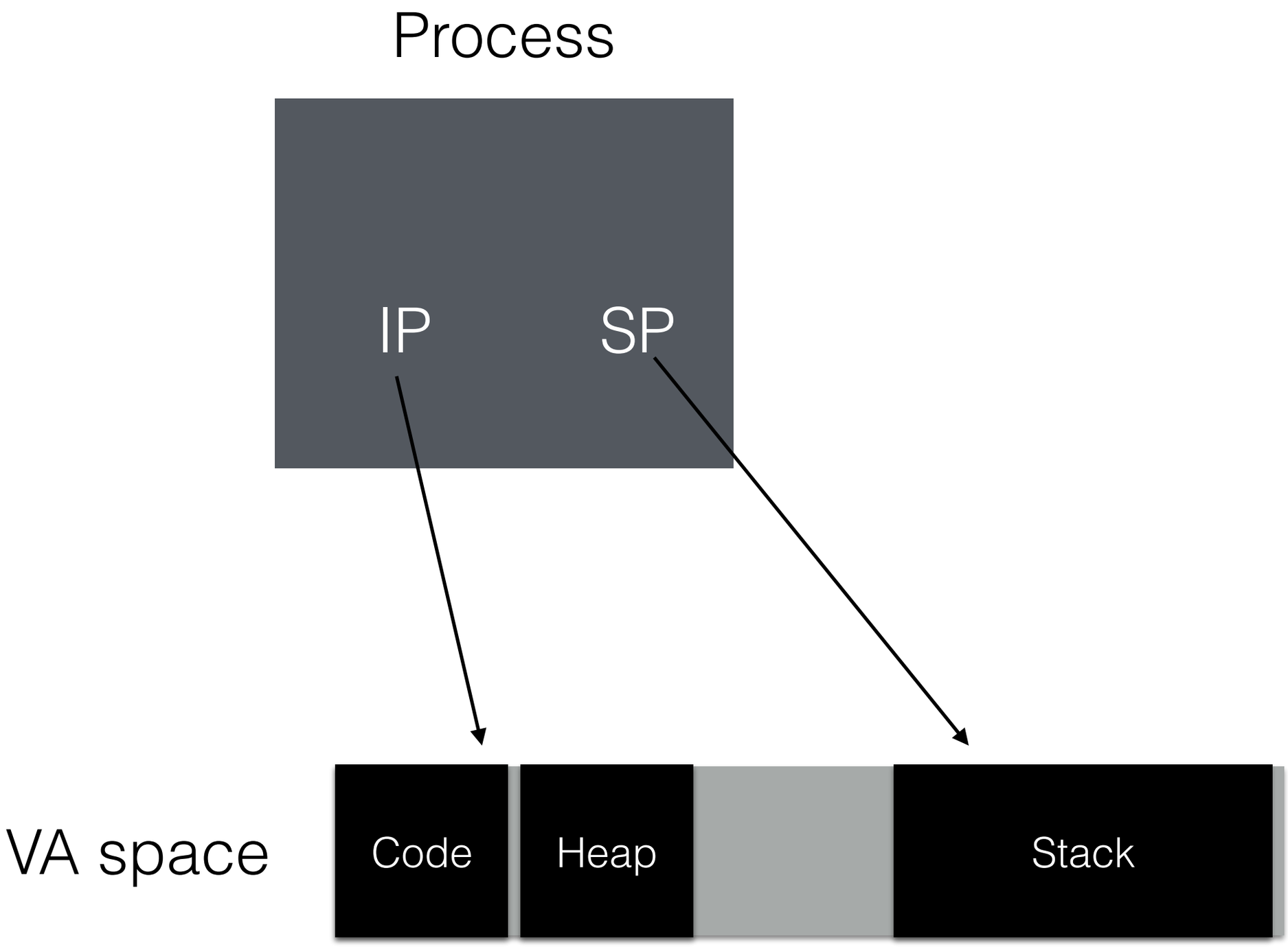
1. Use Threads : just like processes, but, share the address space (i.e. PT)
2. Communicate using common shared data:

Strategy 2

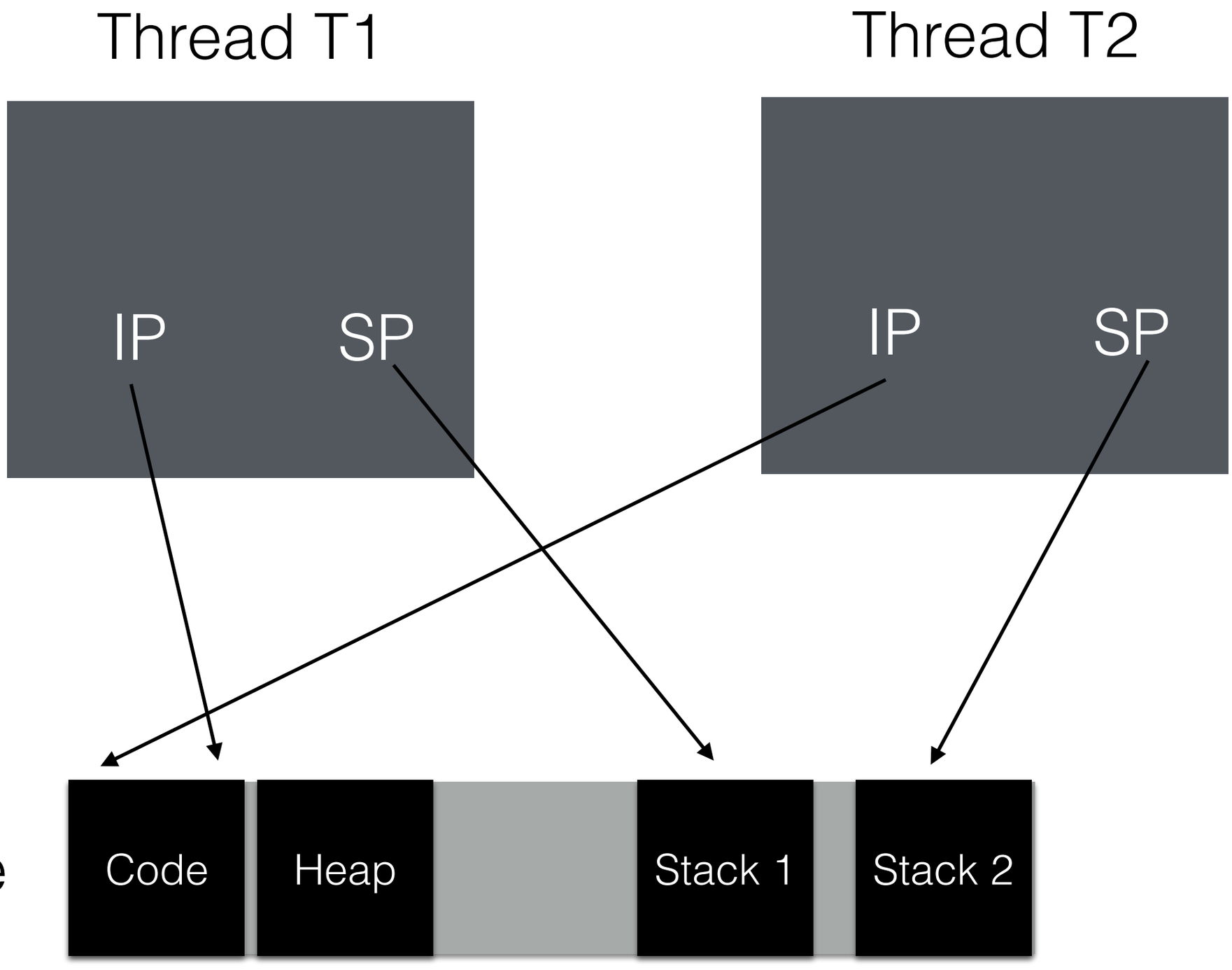
1. Use Threads : just like processes, but, share the address space (i.e. PT)
2. Communicate using common shared data:
3. Single copy of address space!

Threading

Threading



Threading


Thread T1
IP
SP
Thread T2
IP
SP
VA space
Code
Heap
Stack 1
Stack 2

Thread Demos

Thread Demos

1. hello_thread.c

Thread Demos

1. hello_thread.c
2. sum_thread.c - Valgrind it

Thread Demos

1. hello_thread.c
2. sum_thread.c - Valgrind it
    1. valgrind --tool=memcheck ./sum_thread

Thread Demos

1. hello_thread.c
2. sum_thread.c - Valgrind it
    1. valgrind --tool=memcheck ./sum_thread
3. T1.c - working version :)

Thread Demos

1. hello_thread.c
2. sum_thread.c - Valgrind it
    1. valgrind --tool=memcheck ./sum_thread
3. T1.c - working version :)
    1. Disassemble it!

Thread Demos

1. hello_thread.c
2. sum_thread.c - Valgrind it
    1. valgrind --tool=memcheck ./sum_thread
3. T1.c - working version :)
    1. Disassemble it!
    2. objdump -d t1|grep counter

Thread Demos

1. hello_thread.c
2. sum_thread.c - Valgrind it
    1. valgrind --tool=memcheck ./sum_thread
3. T1.c - working version :)
    1. Disassemble it!
    2. objdump -d t1|grep counter
    3. objdump -d t1|grep -C 3 counter|grep -C 3 add

Thread Demos

1. hello_thread.c
2. sum_thread.c - Valgrind it
    1. valgrind --tool=memcheck ./sum_thread
3. T1.c - working version :)
    1. Disassemble it!
    2. objdump -d t1|grep counter
    3. objdump -d t1|grep -C 3 counter|grep -C 3 add

Scheduling Problems …

State

0x20135f: 0
%eax: ?

Thread 1

%eax: ?
IP: cc7

Thread 2

%eax: ?
IP: ?

T1 →

```
cc7:   mov   0x20135f,%eax
ccd:   add   $0x1,%eax
cd0:   mov   %eax,0x20135f
```

Scheduling Problems …

State

0x20135f: 0
%eax: 0

Thread 1

%eax: 0
IP: ccd

Thread 2

%eax: ?
IP: ?

T1

cc7:	mov 0x20135f,%eax
ccd:	add $0x1,%eax
cd0:	mov %eax,0x20135f

Scheduling Problems …

State

0x20135f: 0
%eax: 1

Thread 1

%eax: 1
IP: cd0

Thread 2

%eax: ?
IP: ?

T1

```
cc7:   mov   0x20135f,%eax
ccd:   add   $0x1,%eax
cd0:   mov   %eax,0x20135f
```

Scheduling Problems …

State

0x20135f: 1
%eax: 1

Thread 1

%eax: 1
IP: cd4

Thread 2

%eax: ?
IP: ?

cc7:	mov 0x20135f,%eax
ccd:	add $0x1,%eax
cd0:	mov %eax,0x20135f

T1

Scheduling Problems …

Context Switch

State

0x20135f: 1
%eax: 1

Thread 1

%eax: 1
IP: cd4

Thread 2

%eax: ?
IP: ?

cc7:	mov 0x20135f,%eax
ccd:	add $0x1,%eax
cd0:	mov %eax,0x20135f

T1 →

Scheduling Problems …

Context Switch

State

0x20135f: 1
%eax: 1

Thread 1

%eax: 1
IP: cd4

Thread 2

%eax: ?
IP: ?

cc7:	mov 0x20135f,%eax
ccd:	add $0x1,%eax
cd0:	mov %eax,0x20135f

T1 →

Scheduling Problems …

State

0x20135f: 1
%eax: ?

Thread 1

%eax: 1
IP: cd4

Thread 2

%eax: ?
IP: cc7

T2 →

```
cc7:   mov   0x20135f,%eax
ccd:   add   $0x1,%eax
cd0:   mov   %eax,0x20135f
```

Scheduling Problems …

State

0x20135f: 1
%eax: 1

Thread 1

%eax: 1
IP: cd4

Thread 2

%eax: 1
IP: ccd

T2

```
cc7:   mov   0x20135f,%eax
ccd:   add   $0x1,%eax
cd0:   mov   %eax,0x20135f
```

Scheduling Problems …

State

0x20135f: 1
%eax: 2

Thread 1

%eax: 1
IP: cd4

Thread 2

%eax: 2
IP: cd0

T2 →

cc7:	mov 0x20135f,%eax
ccd:	add $0x1,%eax
cd0:	mov %eax,0x20135f

Scheduling Problems …

State

0x20135f: 2
%eax: 2

Thread 1

%eax: 1
IP: cd4

Thread 2

%eax: 2
IP: cd4

cc7:	mov 0x20135f,%eax
ccd:	add $0x1,%eax
cd0:	mov %eax,0x20135f

T2

Scheduling Problems …(Order #2)

State

0x20135f: 0
%eax: ?

Thread 1

%eax: ?
IP: cc7

Thread 2

%eax: ?
IP: ?

T1 

```
cc7:   mov   0x20135f,%eax
ccd:   add   $0x1,%eax
cd0:   mov   %eax,0x20135f
```

Scheduling Problems …(Order #2)

State

0x20135f: 0
%eax: 0

Thread 1

%eax: 0
IP: ccd

Thread 2

%eax: ?
IP: ?

T1

cc7:	mov 0x20135f,%eax
ccd:	add $0x1,%eax
cd0:	mov %eax,0x20135f

Scheduling Problems …(Order #2)

State

0x20135f: 0
%eax: 1

Thread 1

%eax: 1
IP: cd0

Thread 2

%eax: ?
IP: ?

T1

cc7:	mov 0x20135f,%eax
ccd:	add $0x1,%eax
cd0:	mov %eax,0x20135f

Scheduling Problems …(Order #2)

Context Switch

State

0x20135f: 0
%eax: 1

Thread 1

%eax: 1
IP: cd0

Thread 2

%eax: ?
IP: ?

T1

```
cc7:   mov   0x20135f,%eax
ccd:   add   $0x1,%eax
cd0:   mov   %eax,0x20135f
```

Scheduling Problems …(Order #2)

Context Switch

State

0x20135f: 0
%eax: ?

Thread 1

%eax: 1
IP: cd0

Thread 2

%eax: ?
IP: cc7

T2 →

```
cc7:  mov   0x20135f,%eax
ccd:  add   $0x1,%eax
cd0:  mov   %eax,0x20135f
```

Scheduling Problems …(Order #2)

State

0x20135f: 0
%eax: 0

Thread 1

%eax: 1
IP: cd0

Thread 2

%eax: 0
IP: ccd

T2

cc7:	mov 0x20135f,%eax
ccd:	add $0x1,%eax
cd0:	mov %eax,0x20135f

Scheduling Problems …(Order #2)

State

0x20135f: 0
%eax: 1

Thread 1

%eax: 1
IP: cd0

Thread 2

%eax: 1
IP: cd0

T2 →

cc7:	mov 0x20135f,%eax
ccd:	add $0x1,%eax
cd0:	mov %eax,0x20135f

Scheduling Problems …(Order #2)

State

0x20135f:1
%eax: 1

Thread 1

%eax: 1
IP: cd0

Thread 2

%eax: 1
IP: cd4

cc7:	mov 0x20135f,%eax
ccd:	add $0x1,%eax
cd0:	mov %eax,0x20135f

T2

Scheduling Problems …(Order #2)

Context Switch

State

0x20135f:1
%eax: 1

Thread 1

%eax: 1
IP: cd0

Thread 2

%eax: 1
IP: cd4

cc7:	mov 0x20135f,%eax
ccd:	add $0x1,%eax
cd0:	mov %eax,0x20135f

T2

Scheduling Problems …(Order #2)

Context Switch

State

0x20135f:1
%eax: 1

Thread 1

%eax: 1
IP: cd0

Thread 2

%eax: 1
IP: cd4

T1

```
cc7:  mov   0x20135f,%eax
ccd:  add   $0x1,%eax
cd0:  mov   %eax,0x20135f
```

Scheduling Problems …(Order #2)

Context Switch

State

0x20135f:1
%eax: 1

Thread 1

%eax: 1
IP: cd4

Thread 2

%eax: 1
IP: cd4

cc7:	mov 0x20135f,%eax
ccd:	add $0x1,%eax
cd0:	mov %eax,0x20135f

T1

Timeline View

Thread 1

```
cc7:  mov   0x20135f,%eax
ccd:  add   $0x1,%eax
cd0:  mov   %eax,0x20135f
```

Thread 2

```
cc7:  mov   0x20135f,%eax
ccd:  add   $0x1,%eax
cd0:  mov   %eax,0x20135f
```

Timeline View

Thread 1

```
cc7:  mov   0x20135f,%eax
ccd:  add   $0x1,%eax
cd0:  mov   %eax,0x20135f
```

Thread 2

```
cc7:  mov   0x20135f,%eax
ccd:  add   $0x1,%eax
cd0:  mov   %eax,0x20135f
```

What's the value at 0x20135f?

Timeline View

Thread 1

```
cc7:  mov    0x20135f,%eax
ccd:  add    $0x1,%eax
```

```
cd0:  mov    %eax,0x20135f
```

Thread 2

```
cc7:  mov    0x20135f,%eax
ccd:  add    $0x1,%eax
cd0:  mov    %eax,0x20135f
```

Timeline View

Thread 1

```
cc7:  mov   0x20135f,%eax
ccd:  add   $0x1,%eax
```

```
cd0:  mov   %eax,0x20135f
```

Thread 2

```
cc7:  mov   0x20135f,%eax
ccd:  add   $0x1,%eax
cd0:  mov   %eax,0x20135f
```

What's the value at 0x20135f?

Timeline View

Thread 1

```
cc7: mov    0x20135f,%eax
ccd: add    $0x1,%eax

cd0: mov    %eax,0x20135f
```

Thread 2

```
cc7: mov    0x20135f,%eax
ccd: add    $0x1,%eax

cd0: mov    %eax,0x20135f
```

Timeline View

Thread 1

```
cc7: mov    0x20135f,%eax
ccd: add    $0x1,%eax

cd0: mov    %eax,0x20135f
```

Thread 2

```
cc7: mov    0x20135f,%eax
ccd: add    $0x1,%eax

cd0: mov    %eax,0x20135f
```

What's the value at 0x20135f?

Timeline View

Thread 1

```
cc7: mov    0x20135f,%eax

ccd: add    $0x1,%eax

cd0: mov    %eax,0x20135f
```

Thread 2

```
cc7: mov    0x20135f,%eax

ccd: add    $0x1,%eax

cd0: mov    %eax,0x20135f
```

Timeline View

Thread 1		Thread 2	
cc7:	mov 0x20135f,%eax		
		cc7:	mov 0x20135f,%eax
ccd:	add $0x1,%eax		
		ccd:	add $0x1,%eax
cd0:	mov %eax,0x20135f		
		cd0:	mov %eax,0x20135f

What's the value at 0x20135f?

Timeline View

Thread 1

cc7:	mov 0x20135f,%eax
ccd:	add $0x1,%eax
cd0:	mov %eax,0x20135f

Thread 2

cc7:	mov 0x20135f,%eax
ccd:	add $0x1,%eax
cd0:	mov %eax,0x20135f

Timeline View

Thread 1

cc7:	mov 0x20135f,%eax
ccd:	add $0x1,%eax
cd0:	mov %eax,0x20135f

Thread 2

cc7:	mov 0x20135f,%eax
ccd:	add $0x1,%eax
cd0:	mov %eax,0x20135f

What's the value at 0x20135f?

Timeline View

Thread 1

```
cc7:  mov    0x20135f,%eax
ccd:  add    $0x1,%eax

cd0:  mov    %eax,0x20135f
```

Thread 2

```
cc7:  mov    0x20135f,%eax

ccd:  add    $0x1,%eax
cd0:  mov    %eax,0x20135f
```

Timeline View

Thread 1

```
cc7:  mov    0x20135f,%eax
ccd:  add    $0x1,%eax

cd0:  mov    %eax,0x20135f
```

Thread 2

```
cc7:  mov    0x20135f,%eax
ccd:  add    $0x1,%eax
cd0:  mov    %eax,0x20135f
```

What's the value at 0x20135f?

Debugging

Debugging

1. Concurrency leads to non-determinism

Debugging

1. Concurrency leads to non-determinism
    1. Race conditions

Debugging

1. Concurrency leads to non-determinism
    1. Race conditions
2. Whether bug manifests or not, depends on schedule

Debugging

1. Concurrency leads to non-determinism
    1. Race conditions
2. Whether bug manifests or not, depends on schedule
3. How to program : assume scheduler is malicious!

Atomicity

cc7:	mov 0x20135f,%eax
ccd:	add $0x1,%eax
cd0:	mov %eax,0x20135f

Atomicity

1. Want **all or none** of the following instructions to execute —> atomic instruction

```
cc7:  mov    0x20135f,%eax
ccd:  add    $0x1,%eax
cd0:  mov    %eax,0x20135f
```

Atomicity

1. Want **all or none** of the following instructions to execute —> atomic instruction

```
cc7:  mov    0x20135f,%eax
ccd:  add    $0x1,%eax
cd0:  mov    %eax,0x20135f
```

Atomicity

1. Want **all or none** of the following instructions to execute —> atomic instruction

```
cc7:  mov    0x20135f,%eax
ccd:  add    $0x1,%eax
cd0:  mov    %eax,0x20135f
```

Critical section

Atomicity

1. Want **all or none** of the following instructions to execute —> atomic instruction
2. Want **mutual execution** for **critical section** (if T1 runs, T2 can not, and vice versa)

```
cc7:  mov    0x20135f,%eax
ccd:  add    $0x1,%eax
cd0:  mov    %eax,0x20135f
```

Critical section


MULTITHREADING
THEORY
PRACTICE

Lab Tomorrow

Read Chapter 27 (Thread API) before coming to lab

Practice Questions ...

1. Let's examine a simple program, "loop.s". First, just read and understand it. Then, run it with these arguments (`./x86.py -p loop.s -t 1 -i 100 -R dx`) This specifies a single thread, an interrupt every 100 instructions, and tracing of register `%dx`. What will `%dx` be during the run? Use the `-c` flag to check your answers; the answers, on the left, show the value of the register (or memory value) *after* the instruction on the right has run.
2. Same code, different flags: (`./x86.py -p loop.s -t 2 -i 100 -a dx=3,dx=3 -R dx`) This specifies two threads, and initializes each %dx to 3. What values will `%dx` see? Run with `-c` to check. Does the presence of multiple threads affect your calculations? Is there a race in this code?
3. Run this: `./x86.py -p loop.s -t 2 -i 3 -r -a dx=3,dx=3 -R dx` This makes the interrupt interval small/random; use different seeds (`-s`) to see different interleavings. Does the interrupt frequency change anything?
4. Now, a different program, `looping-race-nolock.s`, which accesses a shared variable located at address 2000; we'll call this variable `value`. Run it with a single thread to confirm your understanding: `./x86.py -p looping-race-nolock.s -t 1 -M 2000` What is `value` (i.e., at memory address 2000) throughout the run? Use `-c` to check.
5. Run with multiple iterations/threads: `./x86.py -p looping-race-nolock.s -t 2 -a bx=3 -M 2000` Why does each thread loop three times? What is final value of `value`?
6. Run with random interrupt intervals: `./x86.py -p looping-race-nolock.s -t 2 -M 2000 -i 4 -r -s 0` with different seeds (`-s 1`, `-s 2`, etc.) Can you tell by looking at the thread interleaving what the final value of `value` will be? Does the timing of the interrupt matter? Where can it safely occur? Where not? In other words, where is the critical section exactly?

Practice Questions …

6. Run with random interrupt intervals: `./x86.py -p looping-race-nolock.s -t 2 -M 2000 -i 4 -r -s 0` with different seeds (`-s 1`, `-s 2`, etc.) Can you tell by looking at the thread interleaving what the final value of `value` will be? Does the timing of the interrupt matter? Where can it safely occur? Where not? In other words, where is the critical section exactly?
7. Now examine fixed interrupt intervals: `./x86.py -p looping-race-nolock.s -a bx=1 -t 2 -M 2000 -i 1` What will the final value of the shared variable `value` be? What about when you change `-i 2`, `-i 3`, etc.? For which interrupt intervals does the program give the "correct" answer?
8. Run the same for more loops (e.g., set `-a bx=100`). What interrupt intervals (`-i`) lead to a correct outcome? Which intervals are surprising?
9. One last program: `wait-for-me.s`. Run: `./x86.py -p wait-for-me.s -a ax=1,ax=0 -R ax -M 2000` This sets the `%ax` register to 1 for thread 0, and 0 for thread 1, and watches `%ax` and memory location 2000. How should the code behave? How is the value at location 2000 being used by the threads? What will its final value be?
10. Now switch the inputs: `./x86.py -p wait-for-me.s -a ax=0,ax=1 -R ax -M 2000` How do the threads behave? What is thread 0 doing? How would changing the interrupt interval (e.g., `-i 1000`, or perhaps to use random intervals) change the trace outcome? Is the program efficiently using the CPU?