Today

- ✓ Sync (MCS)
- ✓ Comm (LRPC)
- ⇒ Sched in SM systems
  - Squillante
  - Fedorova

Friday

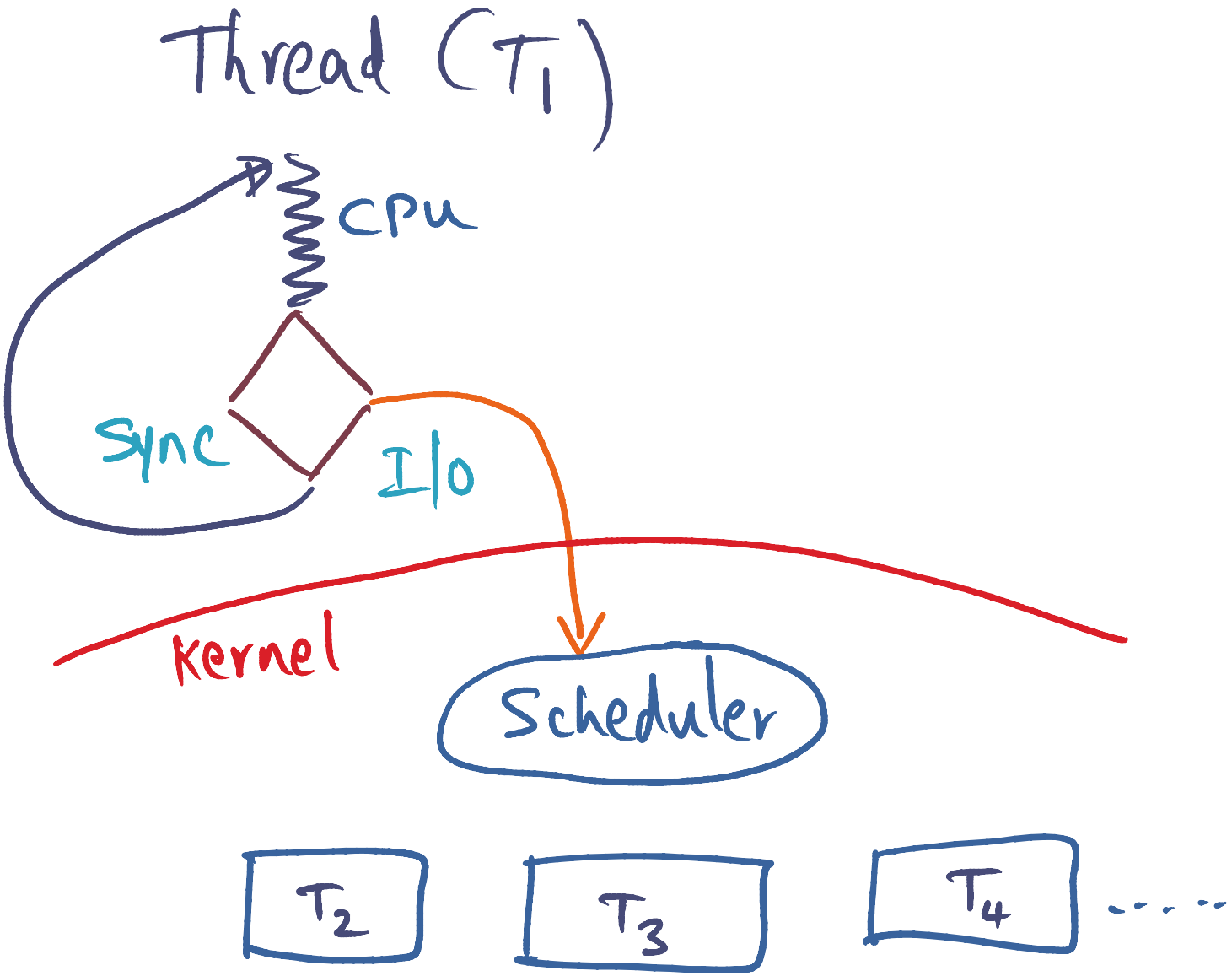
Case Studies of parallel OS

- 
  - Corey
  - Cellular Disco

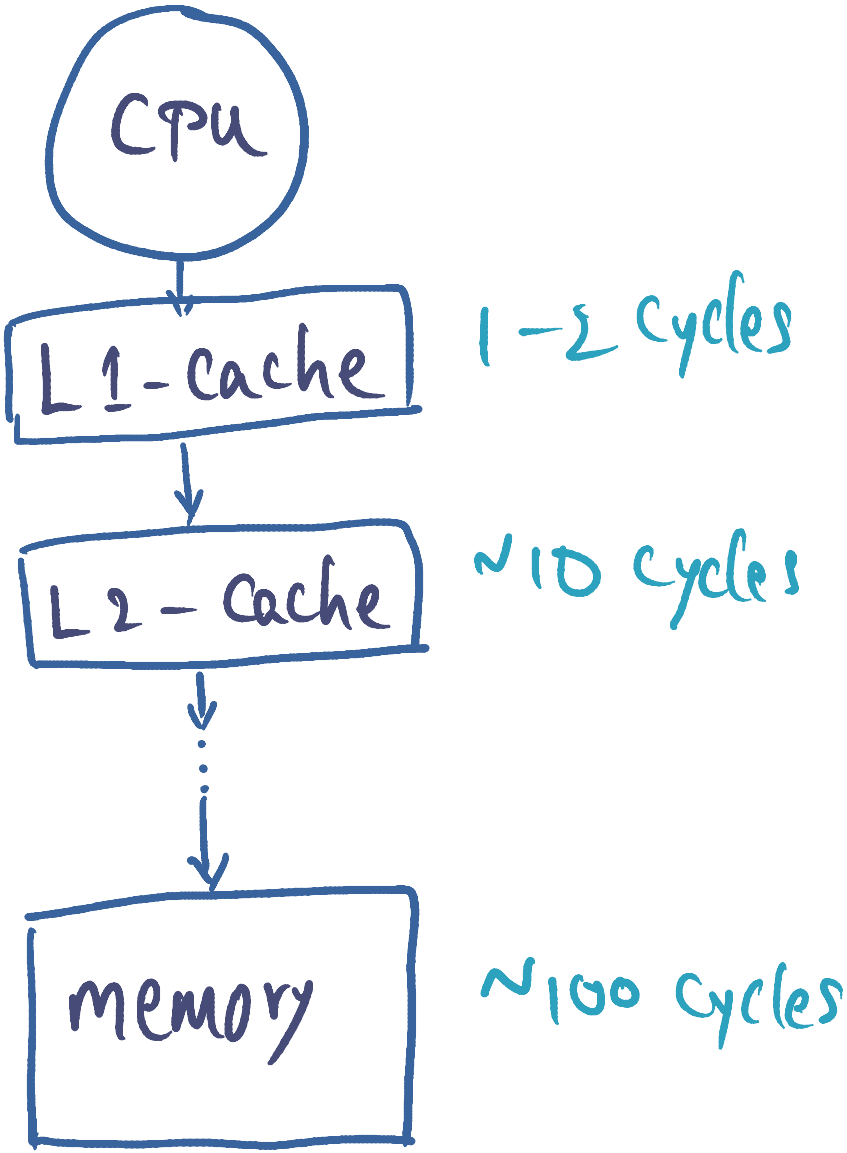
} Partial reading

Please watch video before class

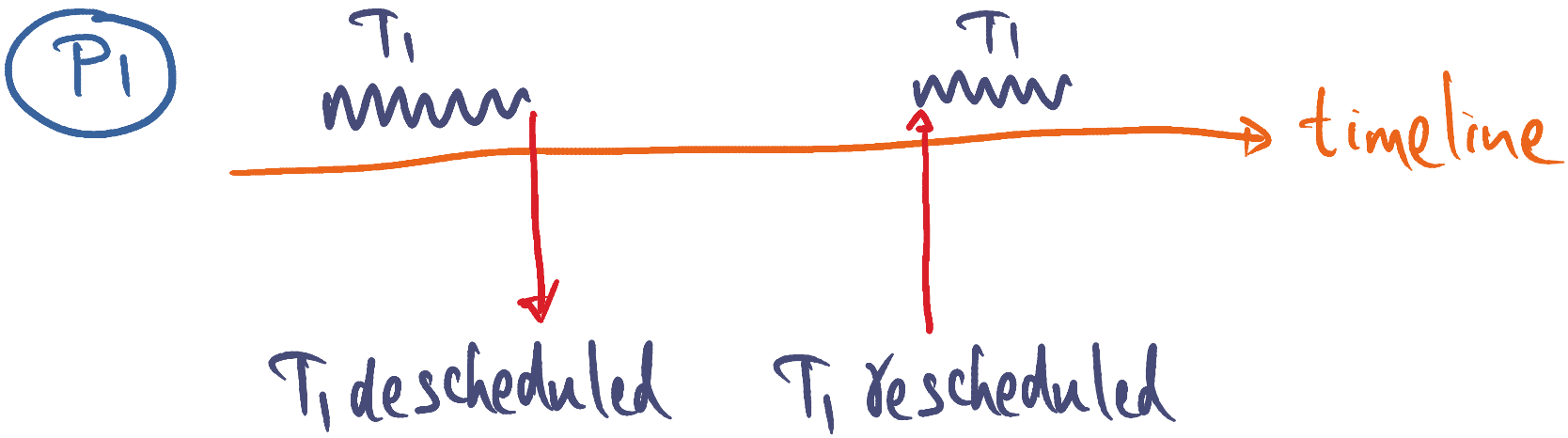
Scheduling – First Principles



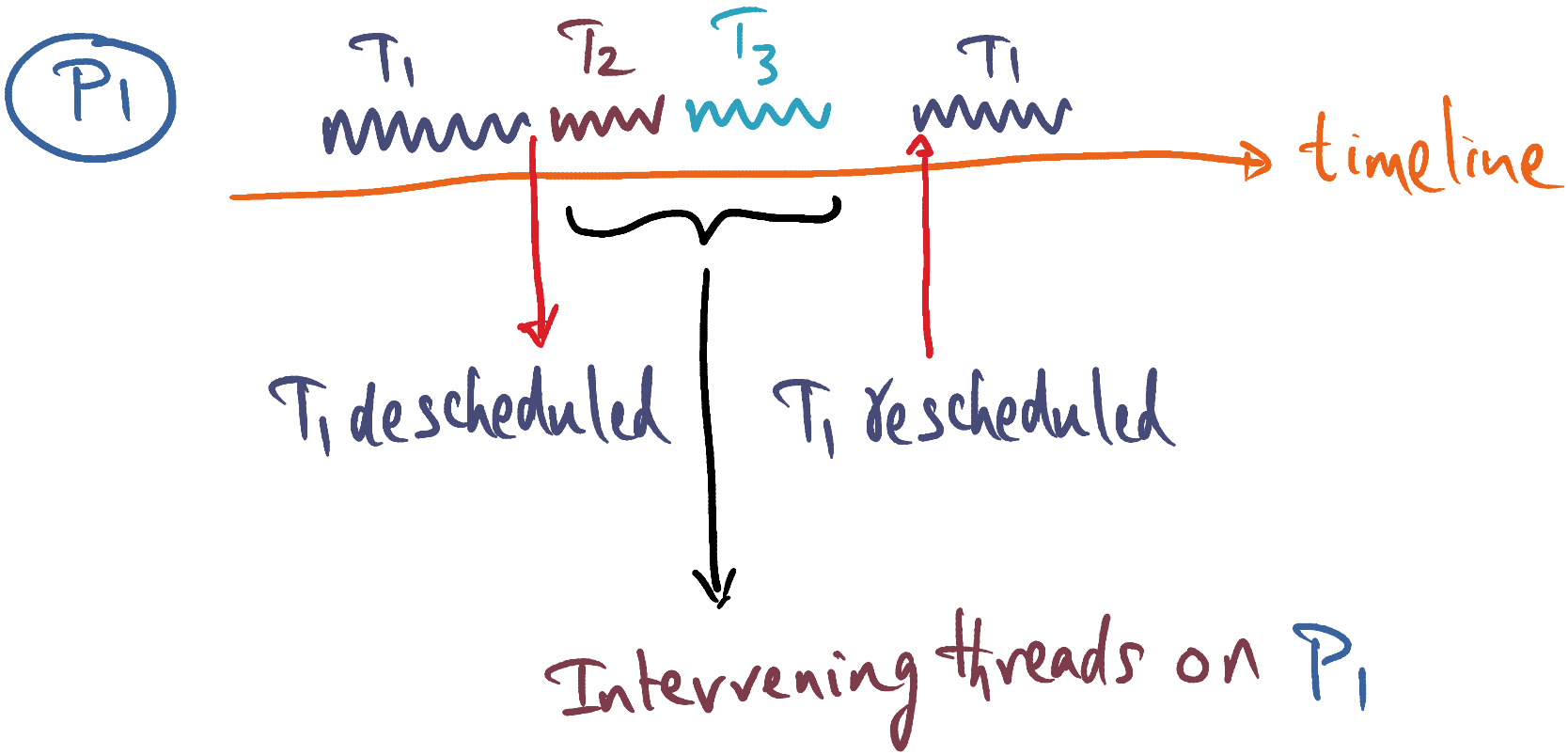
memory hierarchy refresher



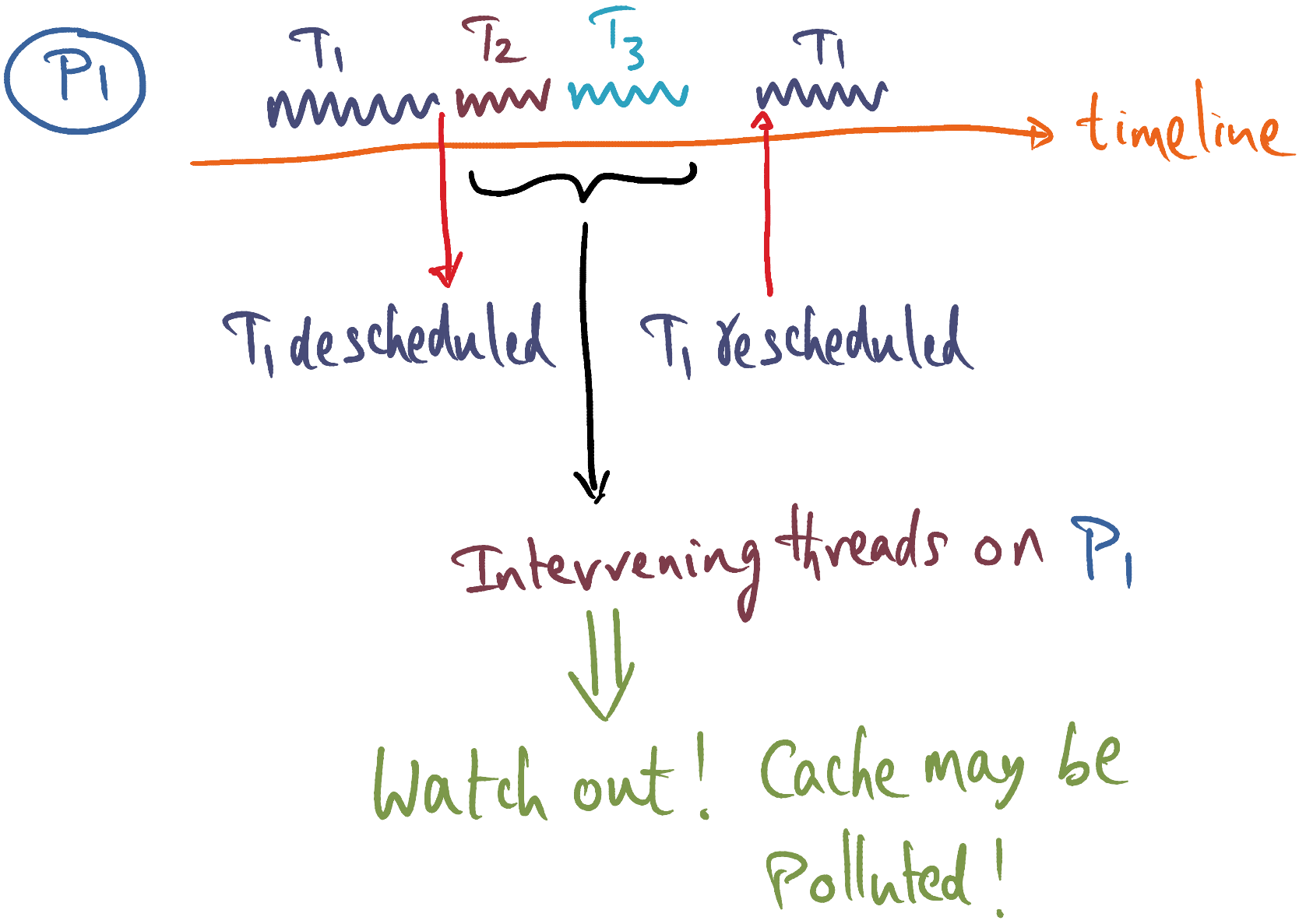
Cache affinity scheduling



Cache affinity scheduling



Cache affinity scheduling



Scheduling Policies

FCFS

Scheduling Policies

FCFS : Ignores affinity for fairness

Scheduling Policies

FCFS : Ignores affinity for fairness

Fixed processor: T_i always on P_{fixed}

Scheduling Policies

FCFS : Ignores affinity for fairness

Fixed processor: T_i always on P_{fixed}

Last processor: T_i on P_{Last}

Scheduling Policies

FCFS: Ignores affinity for fairness

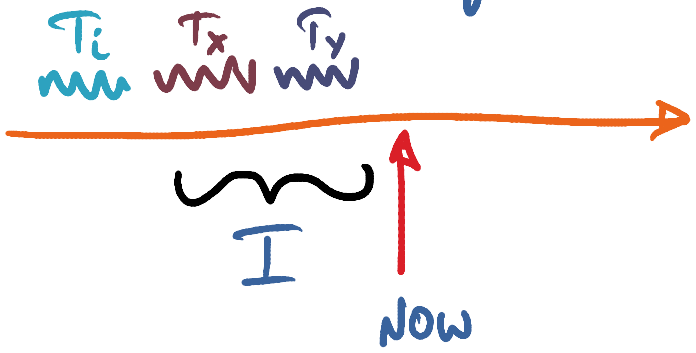
Fixed processor: T_i always on P_{fixed}

Last processor: T_i on P_{Last}

Minimum Intervening: $T_i \rightarrow P_j^{Fmin}$

Minimum Intervening: $T_i \rightarrow P_j^{I_{min}}$

P_j's timeline



Scheduling Policies

FCFS: Ignores affinity for fairness

Fixed processor: T_i always on P_{fixed}

Last processor: T_i on P_{Last}

Minimum Intervening: $T_i \rightarrow P_j^{Fmin}$

Scheduling Policies

FCFS: Ignores affinity for fairness

Fixed processor: T_i always on P_{fixed}

Last processor: T_i on P_{Last}

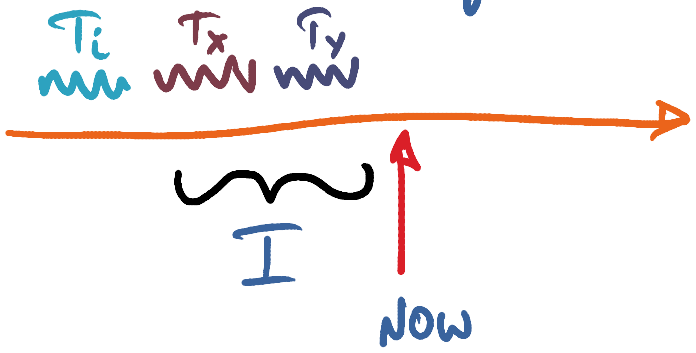
Minimum Intervening: $T_i \rightarrow P_j^{I_{min}}$

Minimum Intervening plus queue:

$T_i \rightarrow P_j^{(I+Q)_{min}}$

Minimum Intervening: $T_i \rightarrow P_j^{I_{min}}$

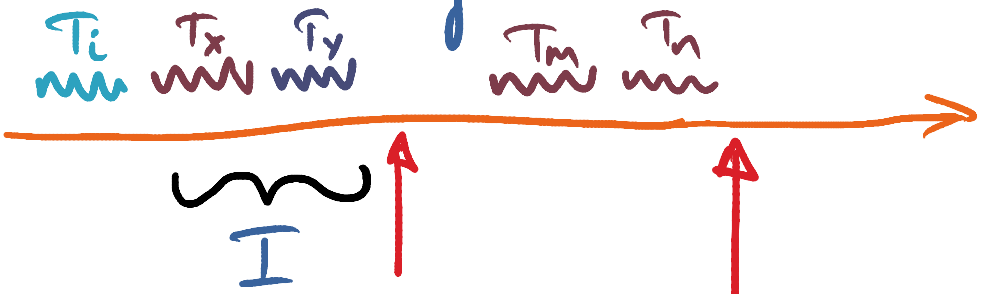
P_j's timeline



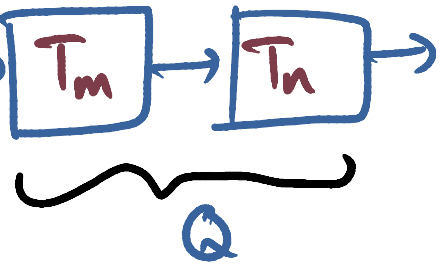
Minimum Intervening plus queue:

$T_i \rightarrow P_j^{(I+Q)_{min}}$

P_j's timeline



P_j's Q



Scheduling Policies

FCFS: Ignores affinity for fairness

Fixed processor: T_i always on P_{fixed}

Last processor: T_i on P_{Last}

Minimum Intervening: $T_i \rightarrow P_j^{I_{min}}$

Minimum Intervening plus queue:

$T_i \rightarrow P_j^{(I+Q)_{min}}$

Scheduling Policies

FCFS: Ignores affinity for fairness

Fixed processor: T_i always on P_{fixed} } focus on cache affinity of T_i

Last processor: T_i on P_{Last} } focus on cache affinity of T_i

Minimum Intervening: $T_i \rightarrow P_j^{I_{min}}$

Minimum Intervening plus queue: $T_i \rightarrow P_j^{(I+Q)_{min}}$

Scheduling Policies

FCFS : Ignores affinity for fairness

Fixed processor: T_i always on P_{fixed}

Last processor: T_i on P_{Last}

} focus on Cache affinity of T_i

Minimum Intervening: $T_i \rightarrow P_j^{I_{min}}$

Minimum Intervening plus queue: $T_i \rightarrow P_j^{(I+Q)_{min}}$

} focus on Cache Pollusion when T_i gets to run

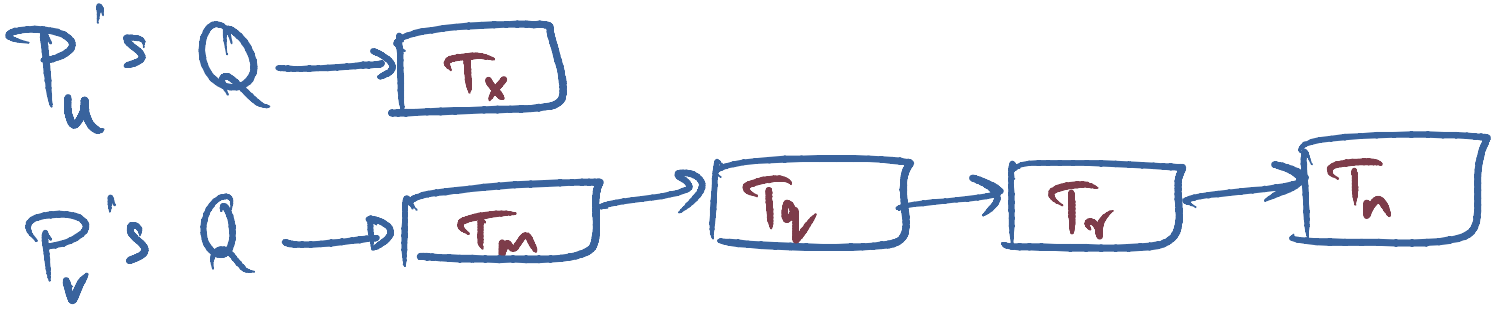
Scheduling Policies

FCFS : Ignores affinity for fairness

Thread Centric
- Fixed processor: T_i always on P_{fixed} — focus on cache affinity of T_i
- Last processor: T_i on P_{Last} — focus on cache affinity of T_i

Processor Centric
- Minimum Intervening: $T_i \rightarrow P_j^{I_{min}}$ — focus on cache pollution when T_i gets to run
- Minimum Intervening plus queue: $T_i \rightarrow P_j^{(I+Q)_{min}}$ — focus on cache pollution when T_i gets to run

Question



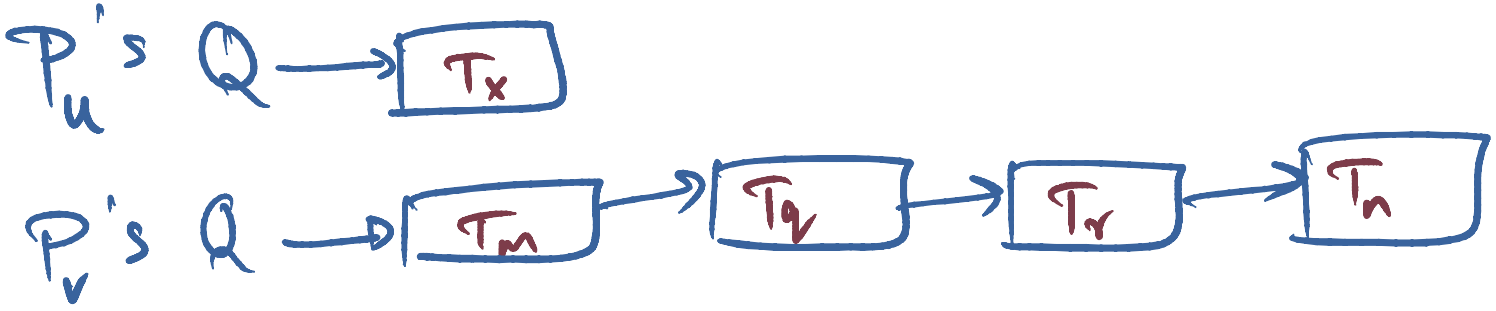
T_y's affinity: $P_u^{\mathcal{I}} = 2$; $P_v^{\mathcal{I}} = 1$ } Num Intervening tasks

Scheduling Policy: Min Inter. plus Q

Which Q for T_y?

☐ P_u's Q ☐ P_v's Q

Question / Solution



T_y's affinity: $P_u^I = 2$; $P_v^I = 1$ } Num Intervening tasks

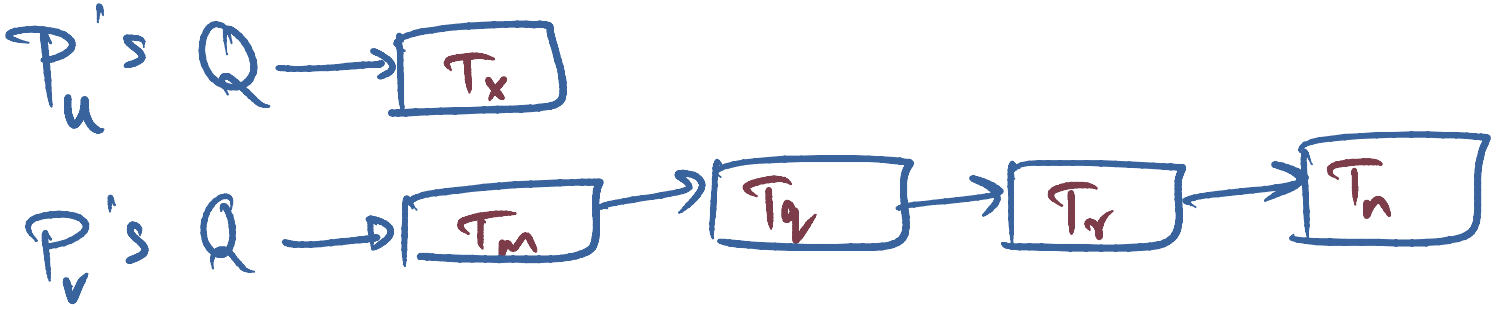
Scheduling Policy : Min Inter. plus q

Which Q for T_y ?

☐ P_u's Q ☐ P_v's Q

$\min_{P_u}^{T_y} = 2 + 1 = 3$

Question / Solution



T_y's affinity: $P_u^I = 2$; $P_v^I = 1$ } Num Intervening tasks

Scheduling Policy : Min Inter. plus Q

Which Q for T_y ?

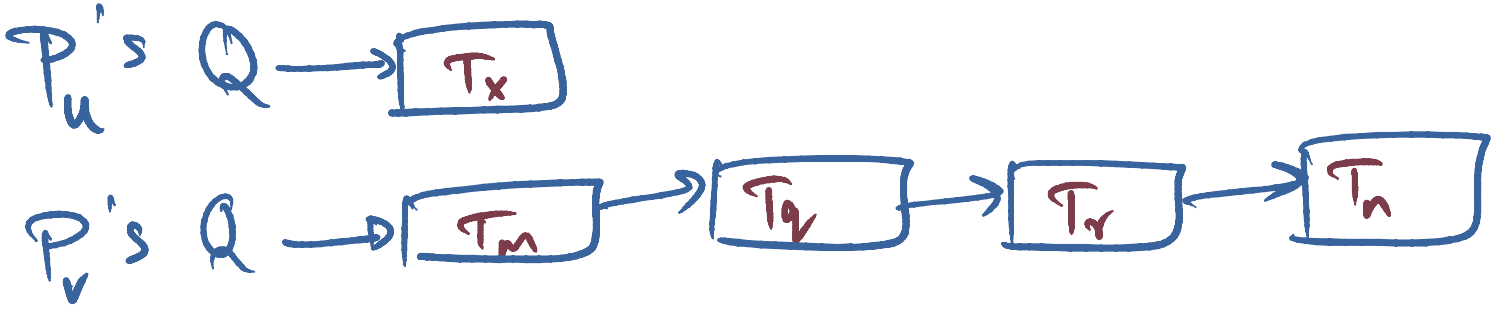
☐ P_u's Q

☐ P_v's Q

$\text{Min}^{T_y}_{P_u} = 2+1 = 3$

$\text{Min}^{T_y}_{P_v} = 1+4 = 5$

Question / Solution



T_y's affinity: $P_u^I = 2$; $P_v^I = 1$ } Num Intervening tasks

Scheduling Policy: Min Inter. plus Q

Which Q for T_y?

☑ P_u's Q

☐ P_v's Q

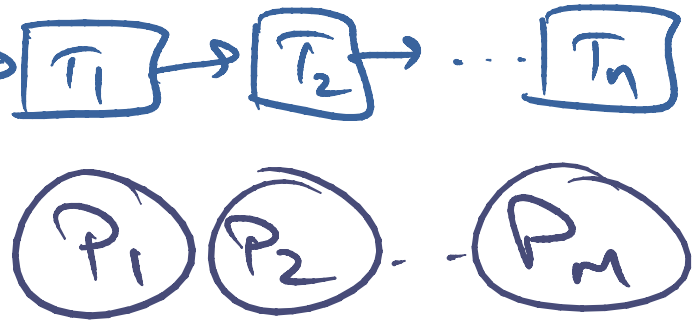
$\text{Min}^{T_y}_{P_u} = 2+1 = 3$

$\text{Min}^{T_y}_{P_v} = 1+4 = 5$

Implementation Issues

Queue Based

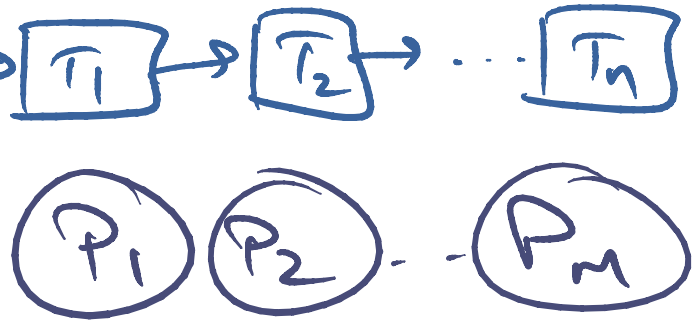
- Global Que



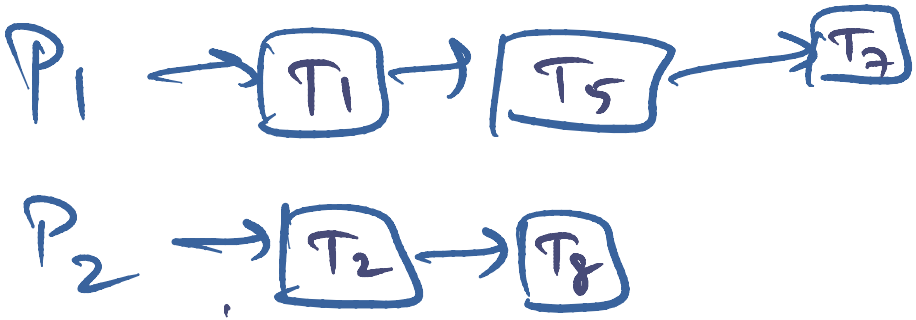
Implementation Issues

Queue Based

- Global Que



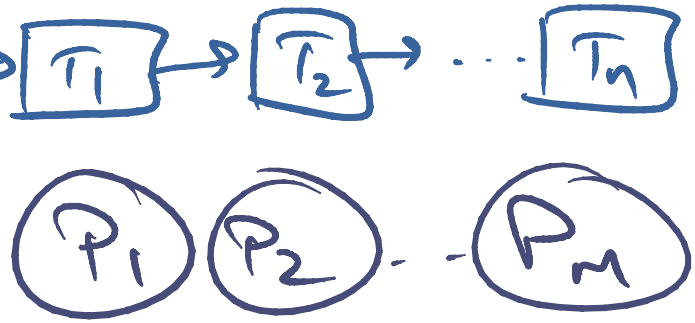
- Affinity-based local queues



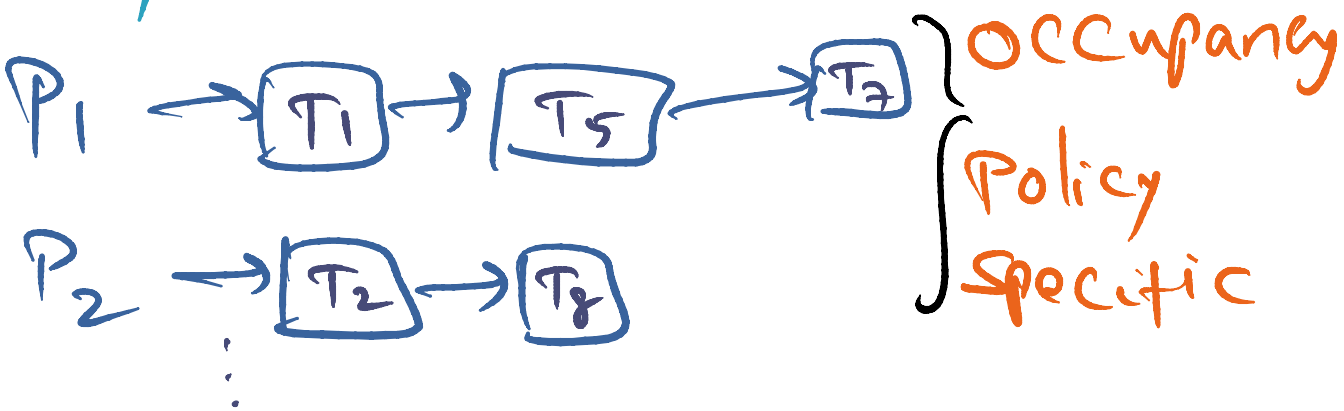
Implementation Issues

Queue Based

- Global Que



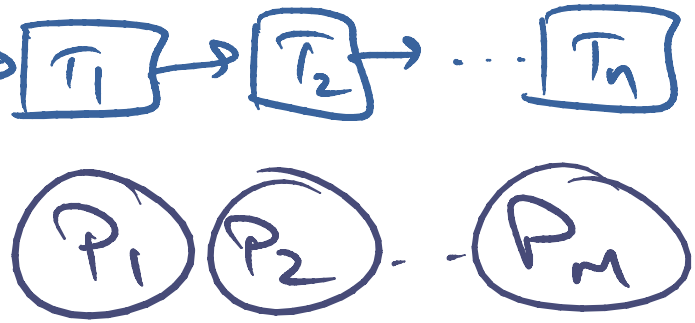
- Affinity-based local queues



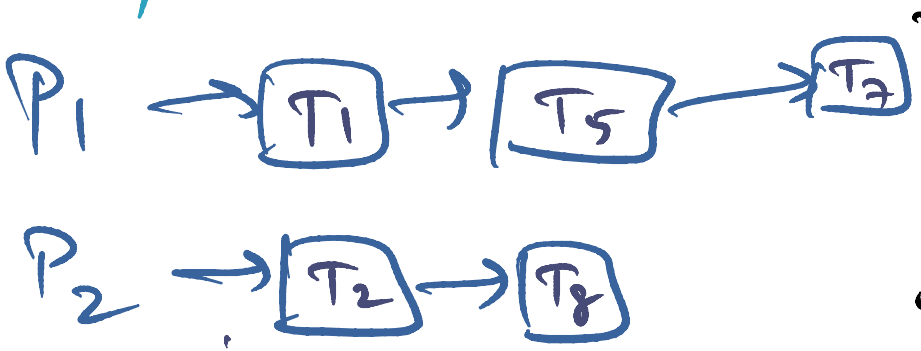
Implementation Issues

Queue Based

- Global Que



- Affinity-based local queues



occupancy

Policy Specific

T_i's Priority $= BP_i + Age_i + Affinity_i$

- Determines position in the queue

Performance

Figures of merit

- Throughput
- Response time
- Variance

Performance

Figures of merit

- Throughput → System Centric
- Response time
- Variance

Performance

Figures of merit

- Throughput → System Centric
- Response time
- Variance

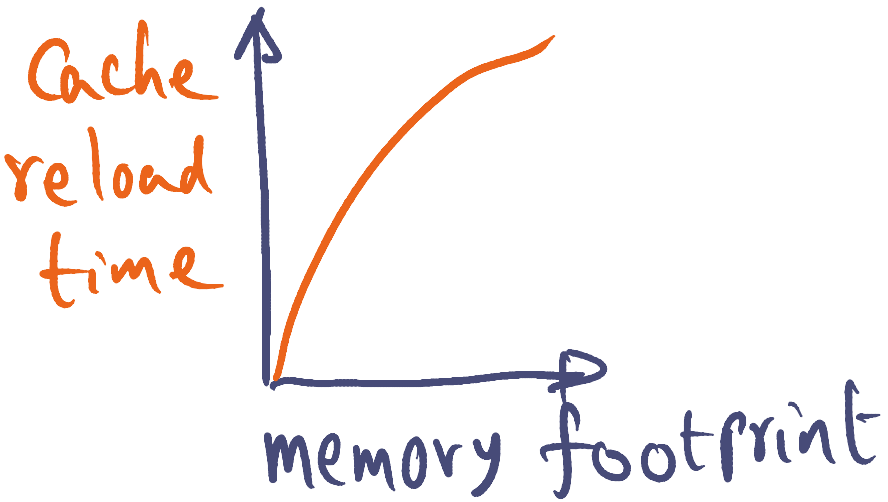
} → User Centric

Performance

Figures of merit

- Throughput → System Centric
- Response time
- Variance

Response time, Variance → User Centric

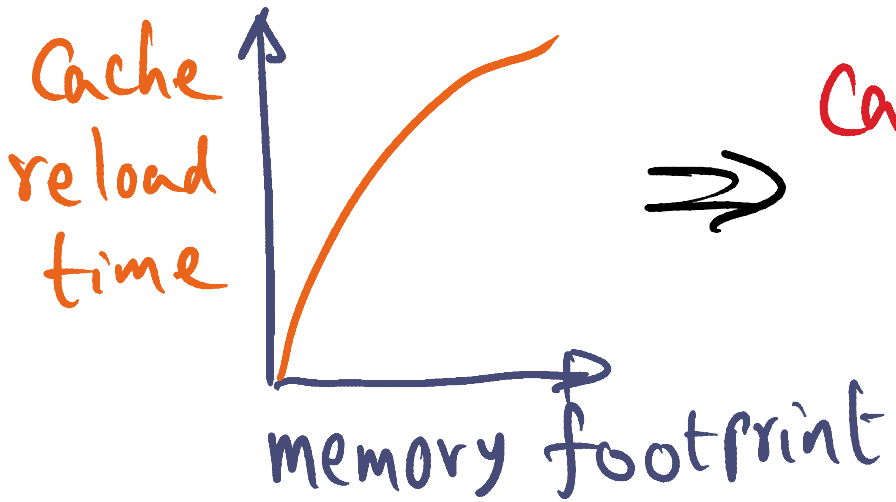


Performance

Figures of merit

- Throughput → System Centric
- Response time
- Variance

→ User Centric



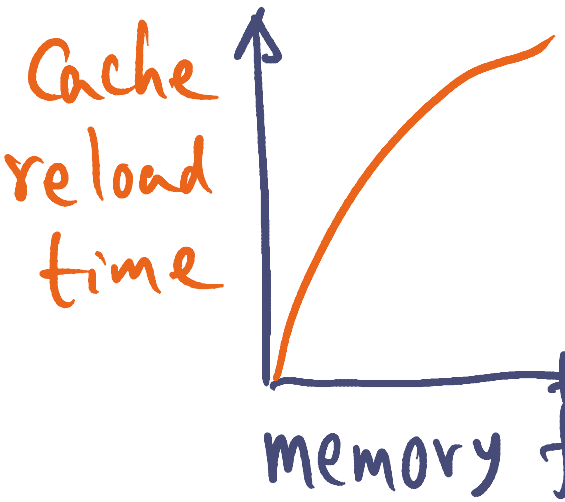
⇒ Cache affinity important

Performance

Figures of merit

- Throughput → System Centric
- Response time
- Variance

→ User Centric



⇒ Cache affinity important

⇓ heavy load

Fixed Processor

Summary

- FCFS is fair but can lead to high variance in response time (small job stuck behind a large job)

Summary

- FCFS is fair but can lead to high variance in response time (small job stuck behind a large job)
- MI and variants good at light load

Summary

- FCFS is fair but can lead to high variance in response time (small job stuck behind a large job)
- MI and variants good at light load
- Heavy load cache is polluted anyway => fixed processor may in fact lead to better perf.

Summary

- FCFS is fair but can lead to high variance in response time (small job stuck behind a large job)
- MI and variants good at light load
- Heavy load cache is polluted anyway => fixed processor may in fact lead to better perf.
- Moral of the story
  - you have to design your scheduler commensurate with the workload you expect

Summary

- FCFS is fair but can lead to high variance in response time (small job stuck behind a large job)
- MI and variants good at light load
- Heavy load cache is polluted anyway => fixed processor may in fact lead to better perf.
- Moral of the story
  - you have to design your scheduler commensurate with the workload you expect
- Inserting idle loop in processor scheduling
  - Good idea?

Summary

- FCFS is fair but can lead to high variance in response time (small job stuck behind a large job)
- MI and variants good at light load
- Heavy load cache is polluted anyway => fixed processor may in fact lead to better perf.
- Moral of the story
  - you have to design your scheduler commensurate with the workload you expect
- Inserting idle loop in processor scheduling
  - Good idea?
  - Waiting for “high affinity” thread

Summary

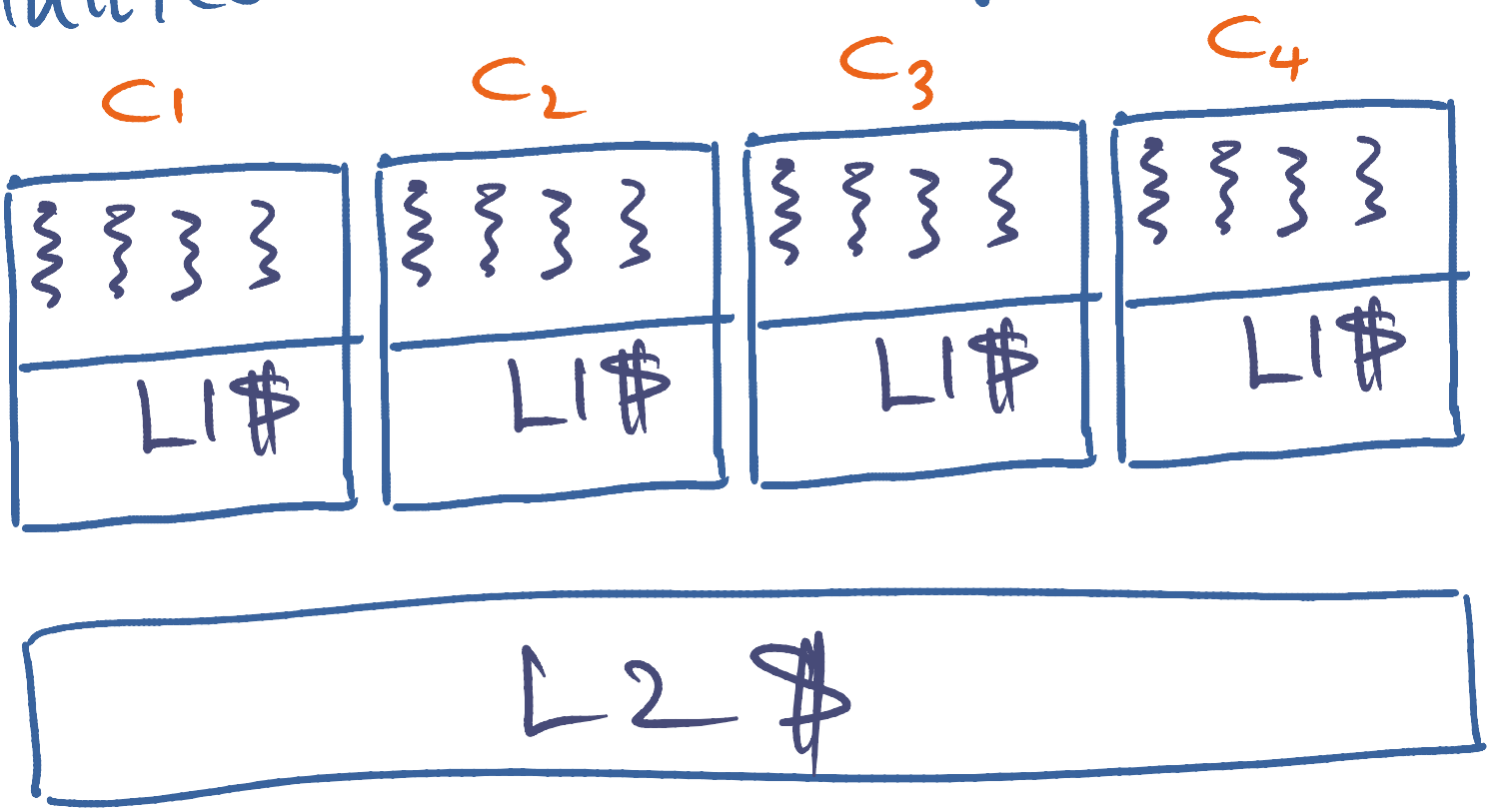
- FCFS is fair but can lead to high variance in response time (small job stuck behind a large job)
- MI and variants good at light load
- Heavy load cache is polluted anyway => fixed processor may in fact lead to better perf.
- Moral of the story
  - you have to design your scheduler commensurate with the workload you expect
- Inserting idle loop in processor scheduling
  - Good idea?
  - Waiting for “high affinity” thread
    - Another instance where “procrastination” may help improve system performance!

Summary

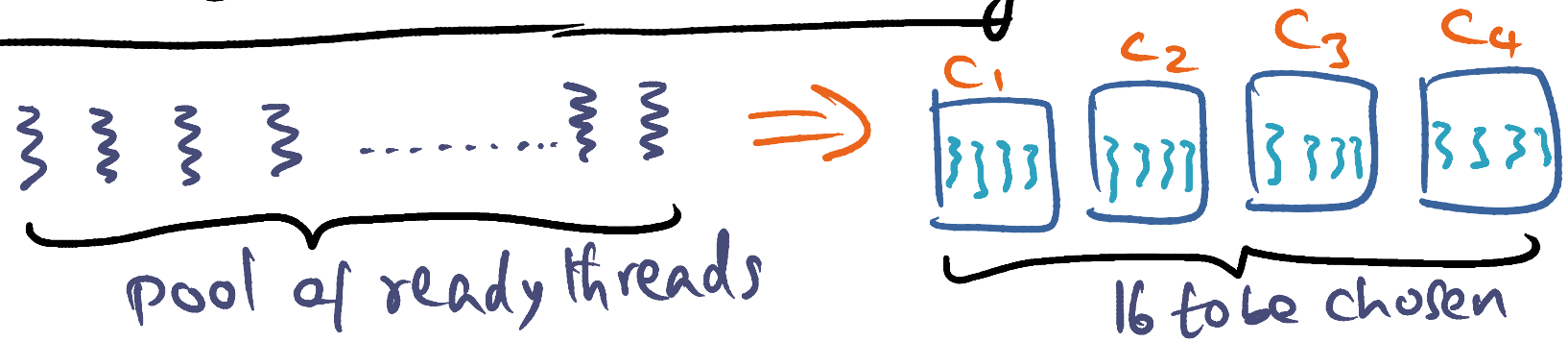
- FCFS is fair but can lead to high variance in response time (small job stuck behind a large job)
- MI and variants good at light load
- Heavy load cache is polluted anyway => fixed processor may in fact lead to better perf.
- Moral of the story
  - you have to design your scheduler commensurate with the workload you expect
- Inserting idle loop in processor scheduling
  - Good idea?
  - Waiting for “high affinity” thread
    - Another instance where “procrastination” may help improve system performance!
      - Have we seen this before?

Cache Affinity and Multicore

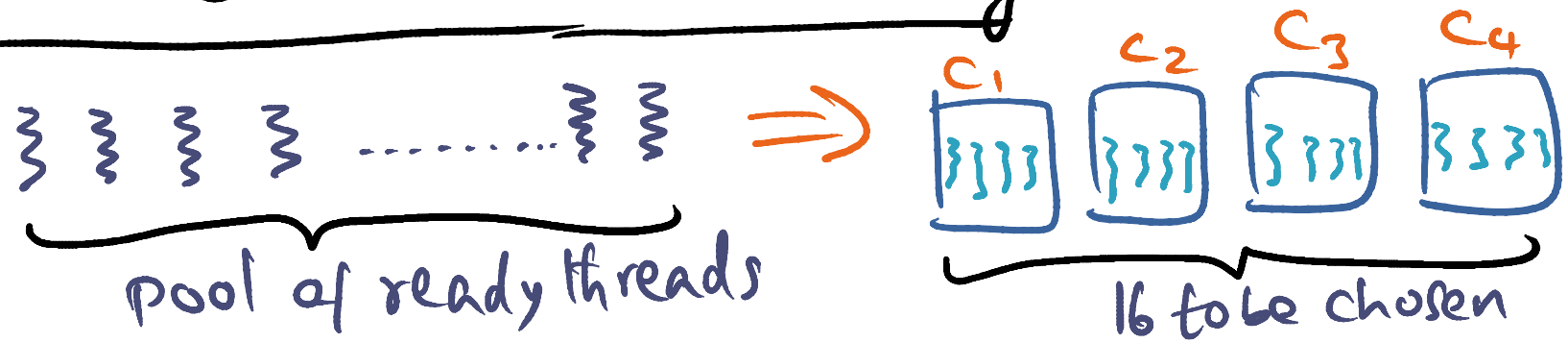
Multicore Multithreaded Processors



Cache Aware Scheduling

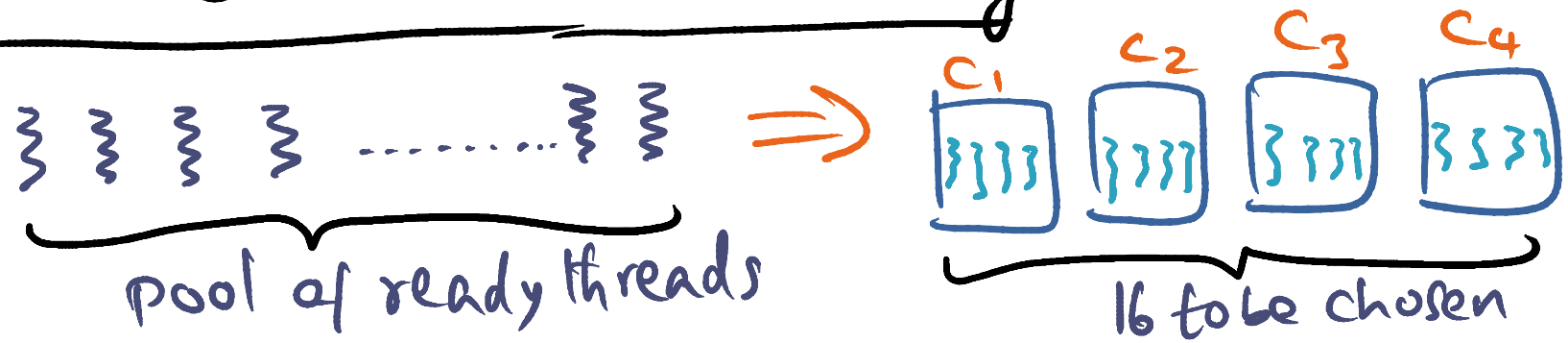


Cache Aware Scheduling



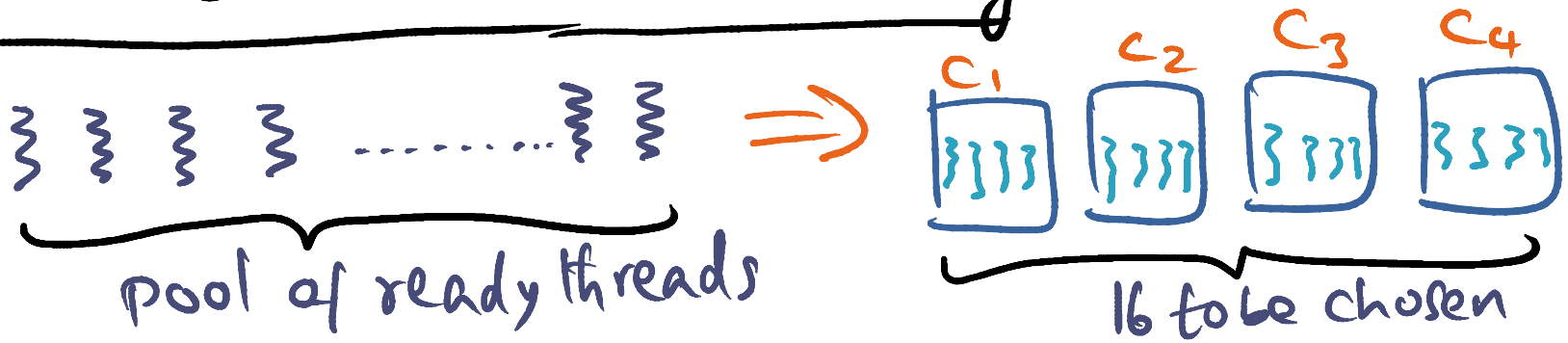
- Cache frugal threads - C_{ft}

Cache Aware Scheduling



- Cache frugal threads – C_{ft}
- Cache hungry threads – C_{ht}

Cache Aware Scheduling



- Cache frugal threads – C_{ft}
- Cache hungry threads – C_{ht}

$$\sum_{1}^{n} C_{ft} + \sum_{1}^{m} C_{ht} < \text{Size(L2 \$)}$$

$$m+n = 16$$

Friday

Case Studies of parallel OS

* 

* Corey
* Cellular Disco

} Partial reading

Please watch video before class

Next week

Distributed systems

* Watch Lamport clock video
  (First video in Lesson 5 on Udacity)