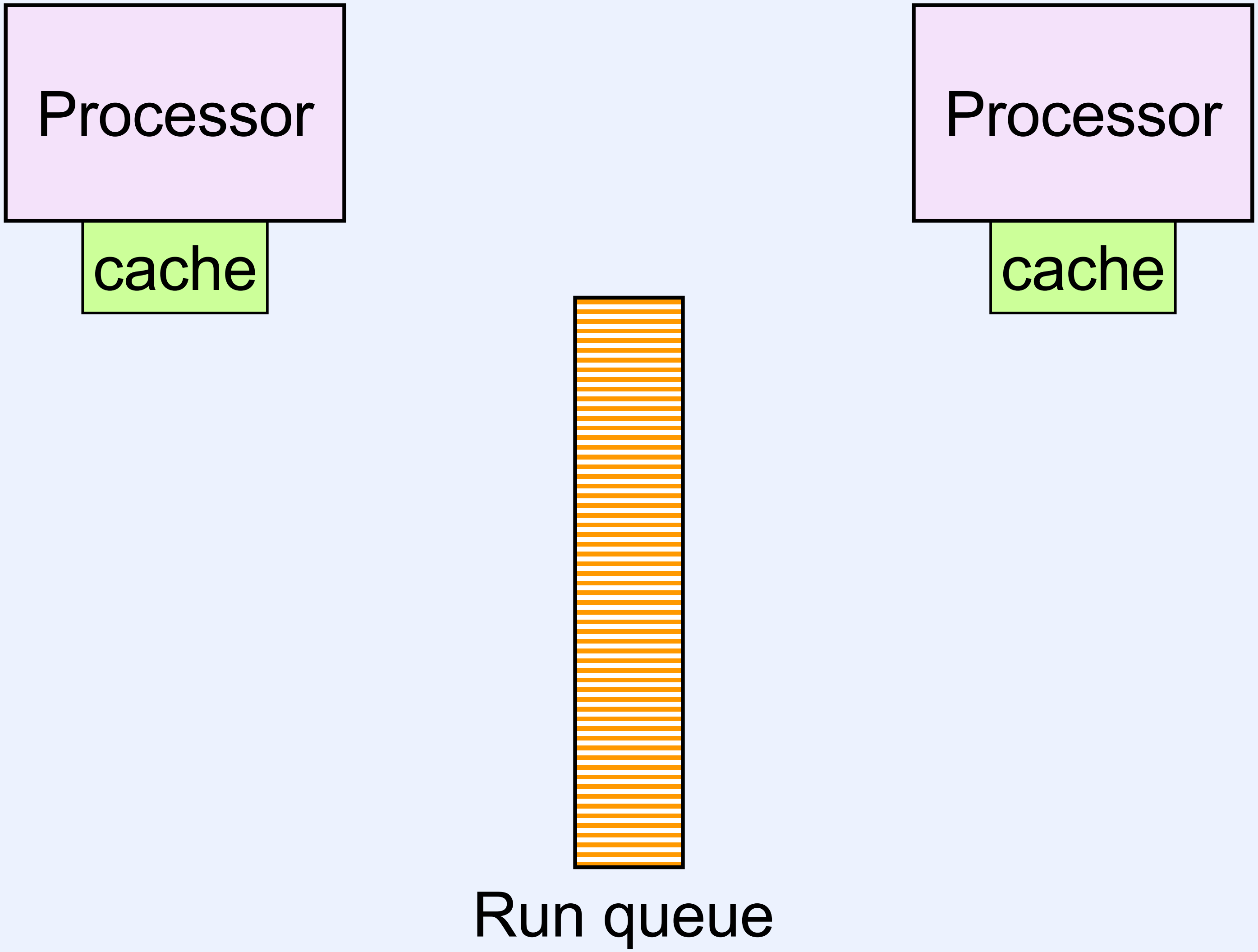
Scheduling Part 3

Diagram



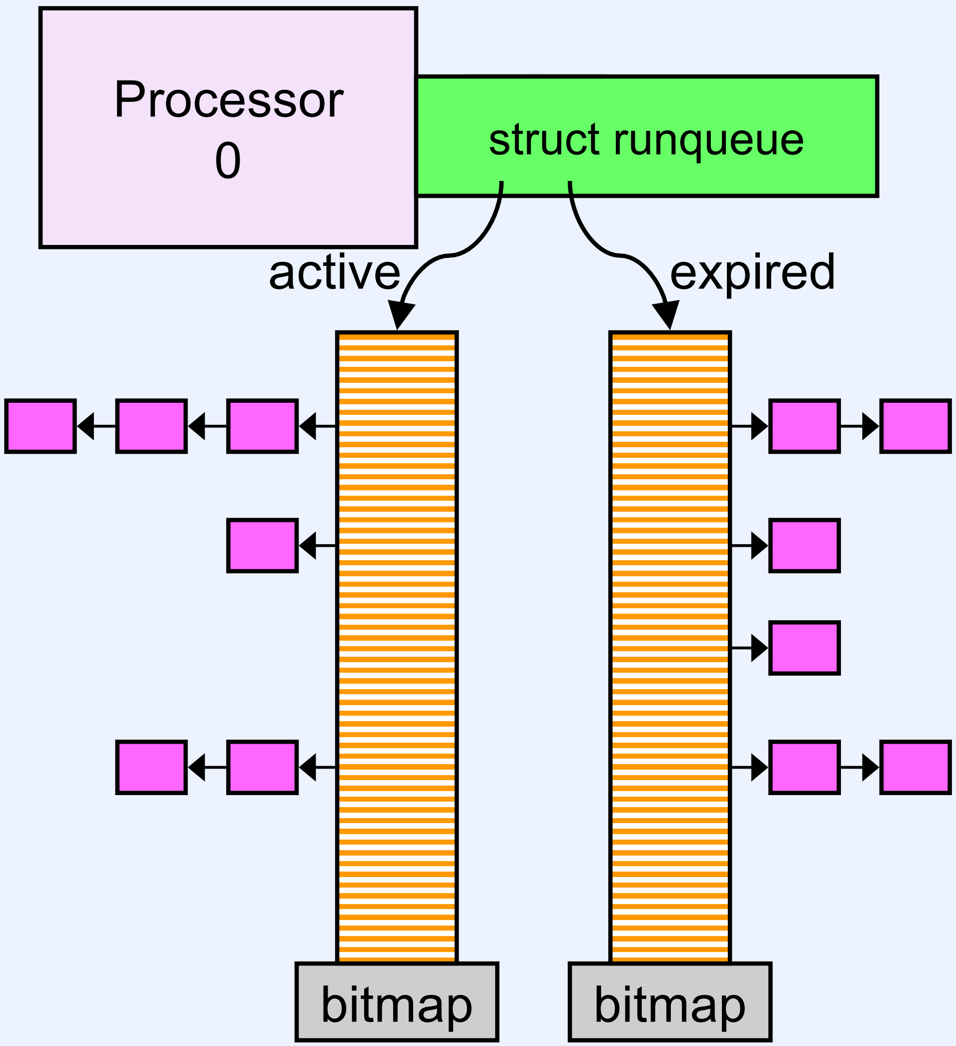
Old Scheduler: Problems

- O(n) execution
- Poor interactive performance with heavy loads
- SMP contention for run-queue lock
- SMP affinity
  - cache “footprint”

O(1) Scheduler

- All concerns of old scheduler plus:
  - efficient, scalable execution
  - identify and favor interactive processes
  - good SMP performance
    - minimal lock overhead
    - processor affinity

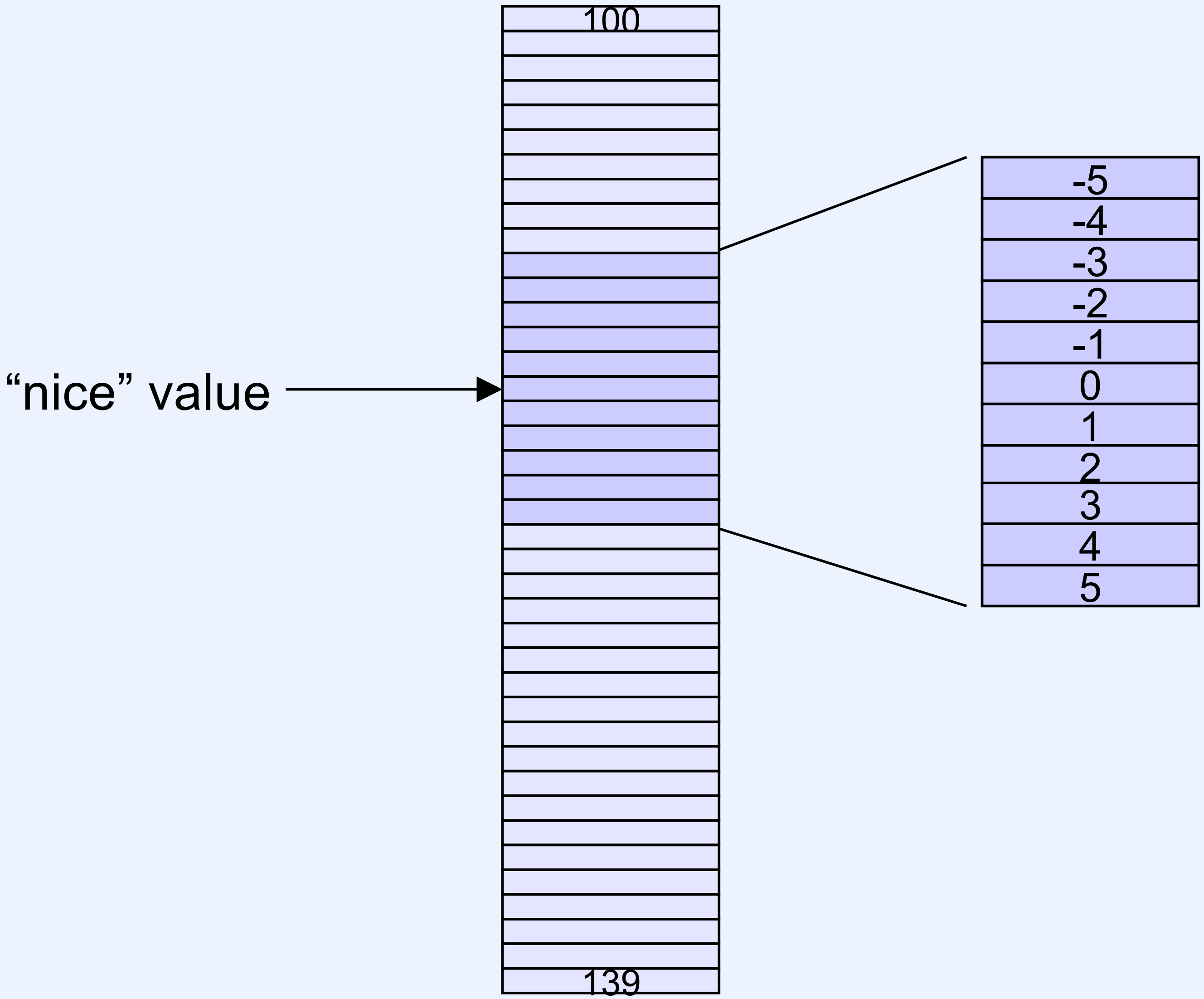
O(1) Scheduler: Data Structures



O(1) Scheduler: Queues

- Two queues per processor
  - active: processes with remaining time slice
  - expired: processes whose time slices expired
  - each queue is an array of lists of processes of the same priority
    - bitmap indicates which priorities have processes
  - processors scheduled from private queues
    - infrequent lock contention
    - good affinity

O(1) Scheduler: Priorities



O(1) Scheduler: Actions

- Process switch
  - pick best priority from active queue
    - if empty, switch active and expired
  - new process's time slice is function of its priority
- Wake up
  - priority is boosted or dropped depending on sleep time
  - interactive processes are defined as those whose priority is above a certain threshold
- Time-slice expiration
  - normal processes join expired queue
  - real-time join active queue

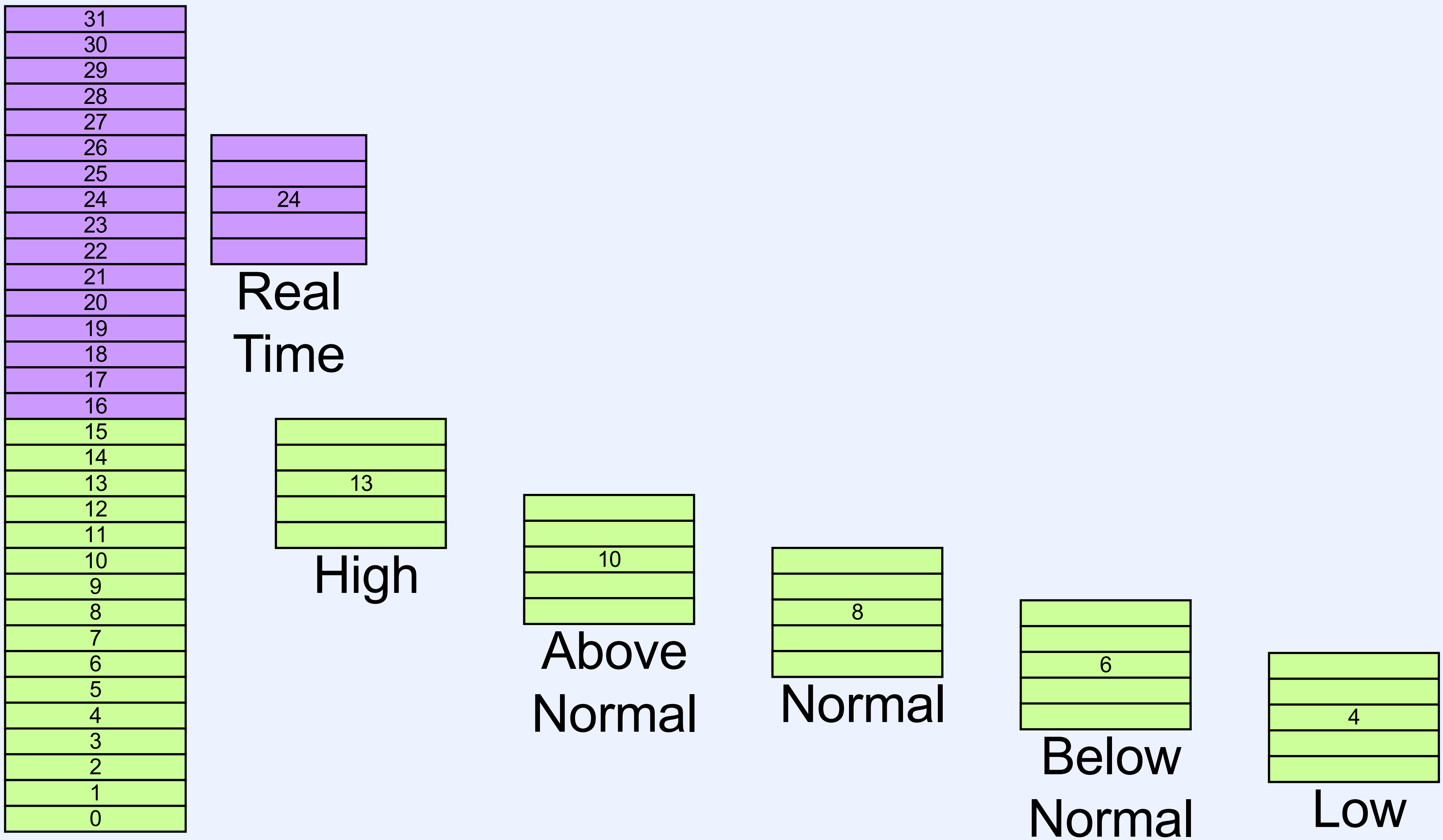
O(1) Scheduler: Load Balancing

- Processors with empty queues steal from busiest processor
  - checked every millisecond
- Processors with relatively small queues also steal from busiest processor
  - checked every 250 milliseconds

Scheduling in Windows

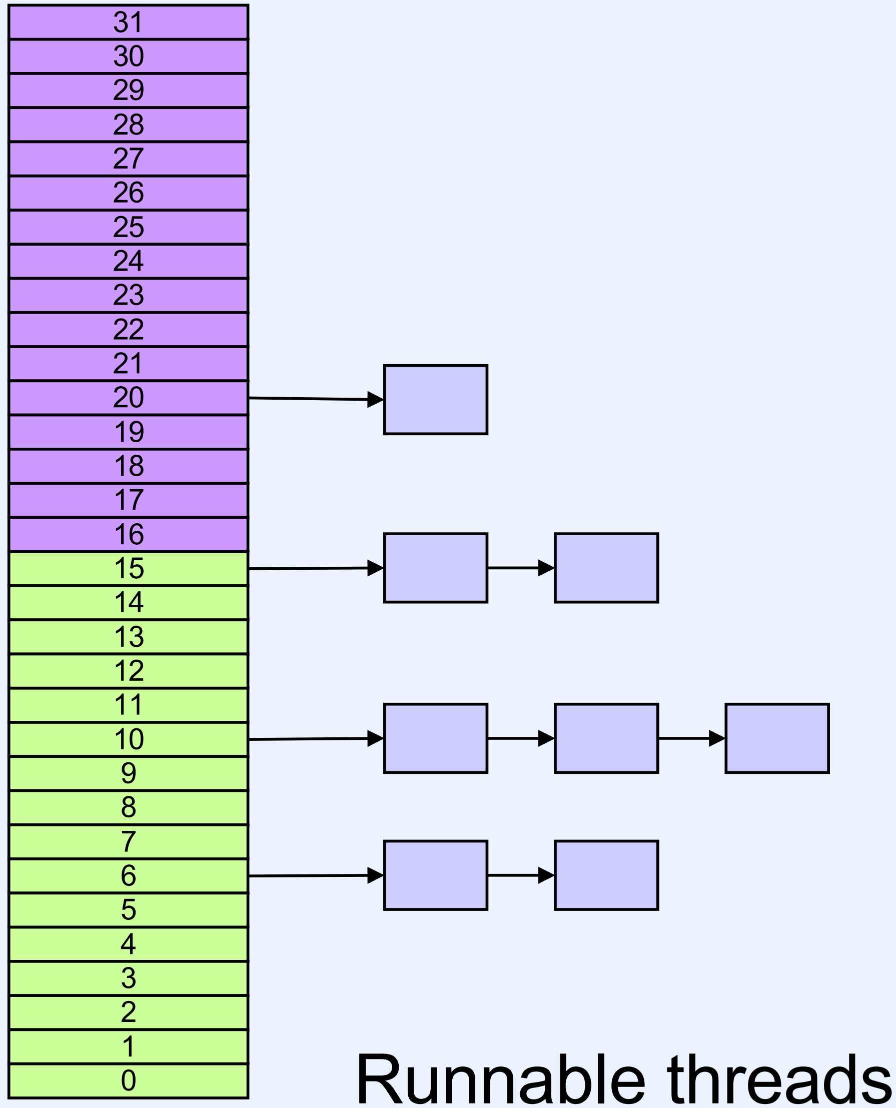
- Handling “normal” interactive and compute-bound threads
- Real-time threads
- Multiple processors

Priorities



31
30
29
28
27
26
25
24
23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
24
Real Time
13
High
10
Above Normal
8
Normal
6
Below Normal
4
Low

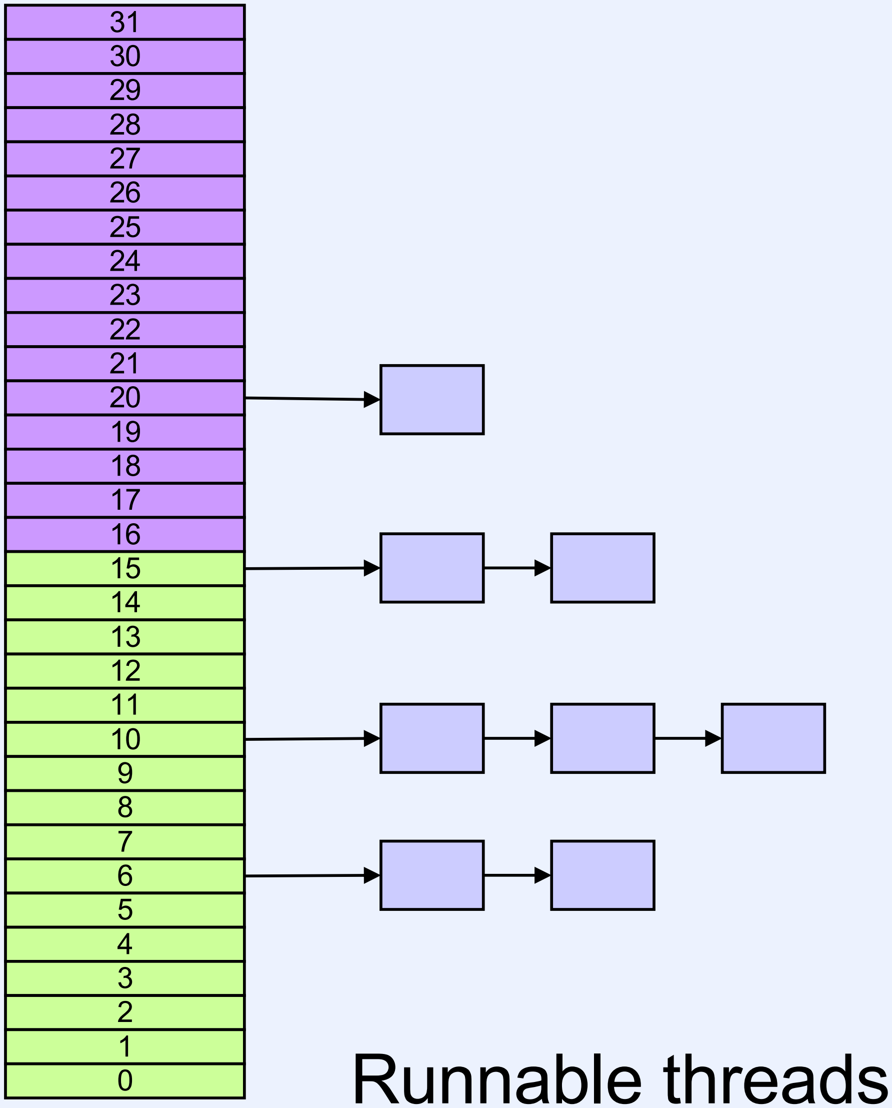
Uniprocessor Windows

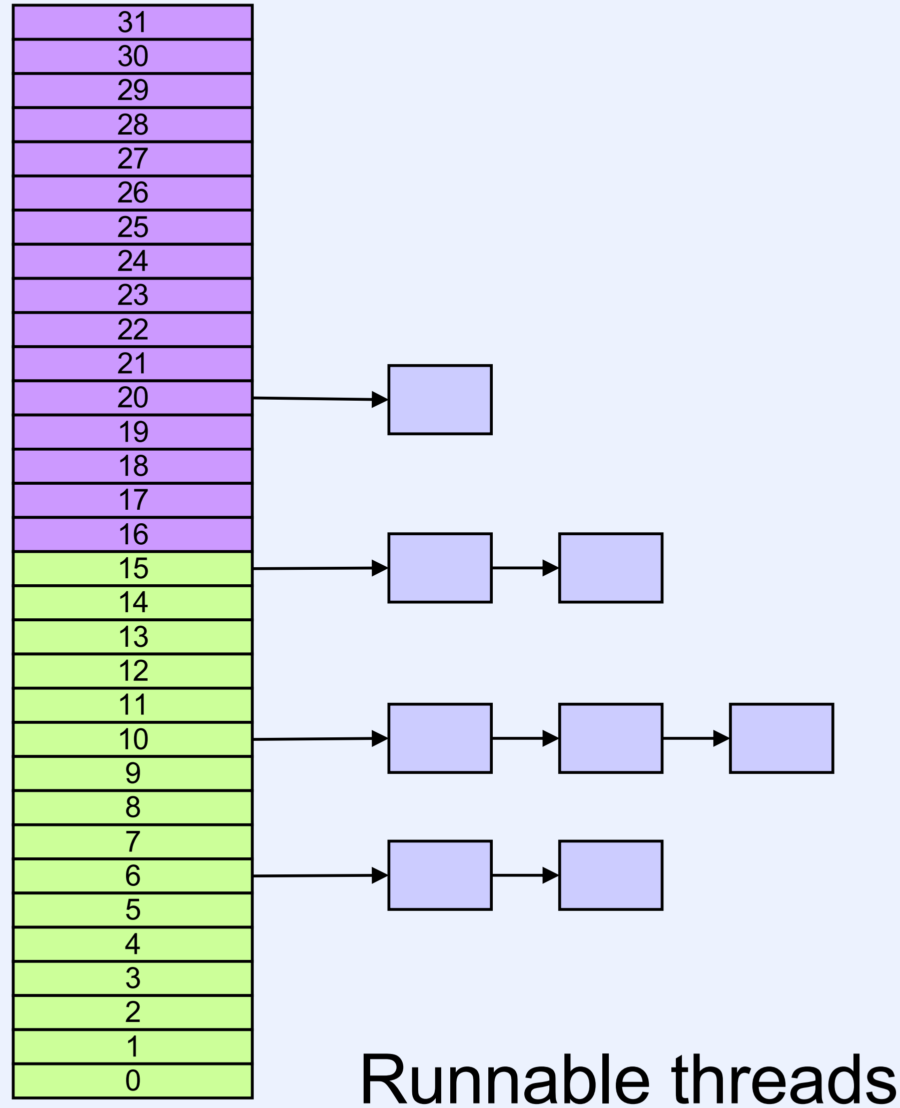


Improving Real Time

- Multimedia applications need 80% of processor time
- Make sure normal applications get at least 20%
- How?
- Windows solution: MMCSS
  – multimedia class scheduler service
  – dynamically manage multimedia threads
    - run at real-time priority 80% of time
    - run at normal priority 20% of time

Multiprocessor Windows

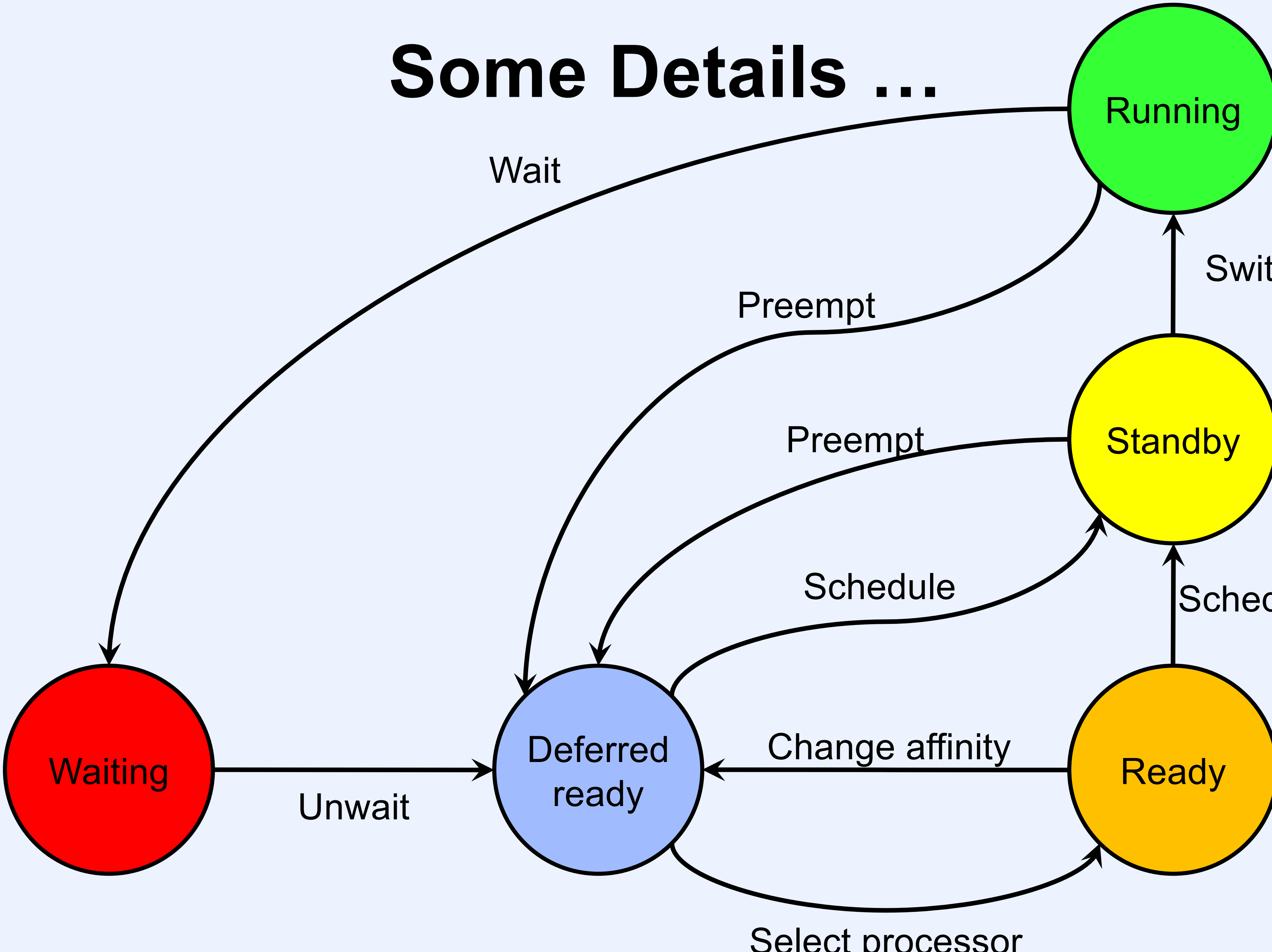




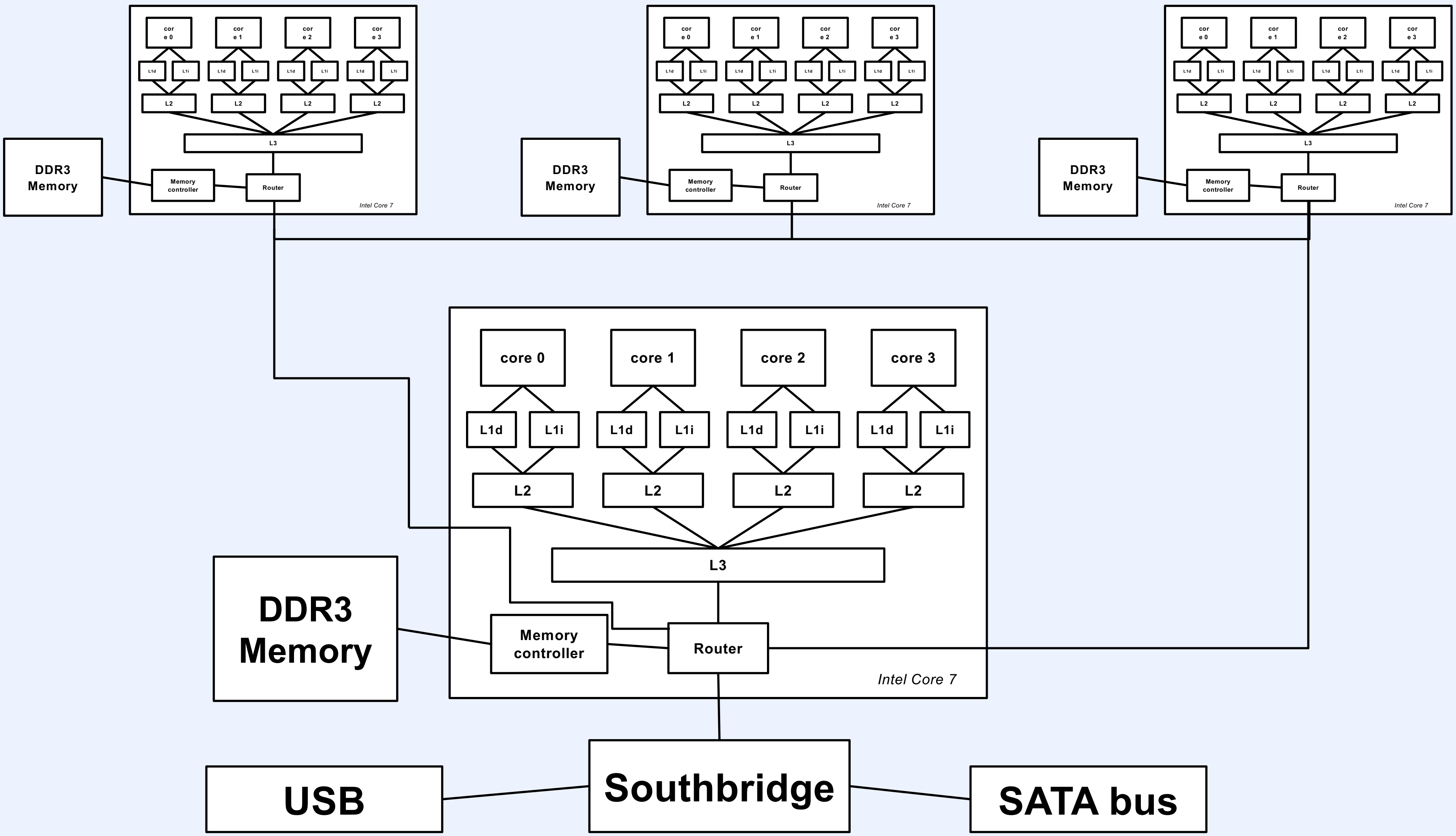
Which Processor?

- Newly created thread assigned *ideal processor*
  - randomly chosen
- May also set *affinity mask*
  - may be scheduled only on processors in mask
- Scheduling decision:
  - if idle processors available
    - first preference: ideal processor (if idle)
    - second preference: most recent processor (if idle)
  - otherwise
    - joins run queue of ideal processor

Some Details …



NUMA

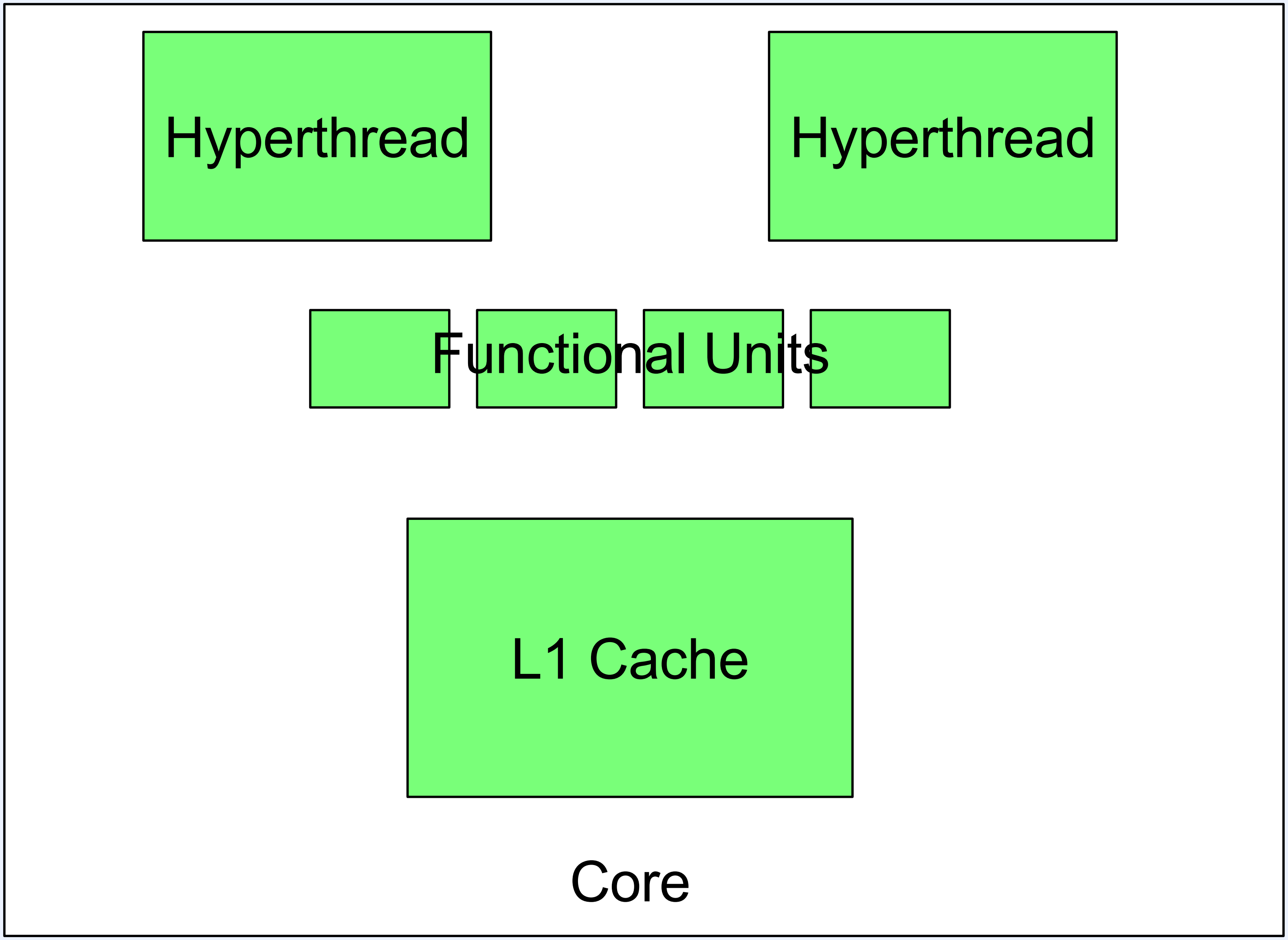


DDR3 Memory
Memory controller
Router
L3
L2
L1d
L1i
core 0
core 1
core 2
core 3
Intel Core 7
USB
Southbridge
SATA bus

Scheduling Concerns

- Hyperthreads
  - two instruction streams sharing same functional units and same L1 cache
- How long does cache footprint matter?
  - what cache parameters are important?
- When is it a good idea to put a thread on:
  - a different core?
  - a different NUMA node?

Hyperthreads



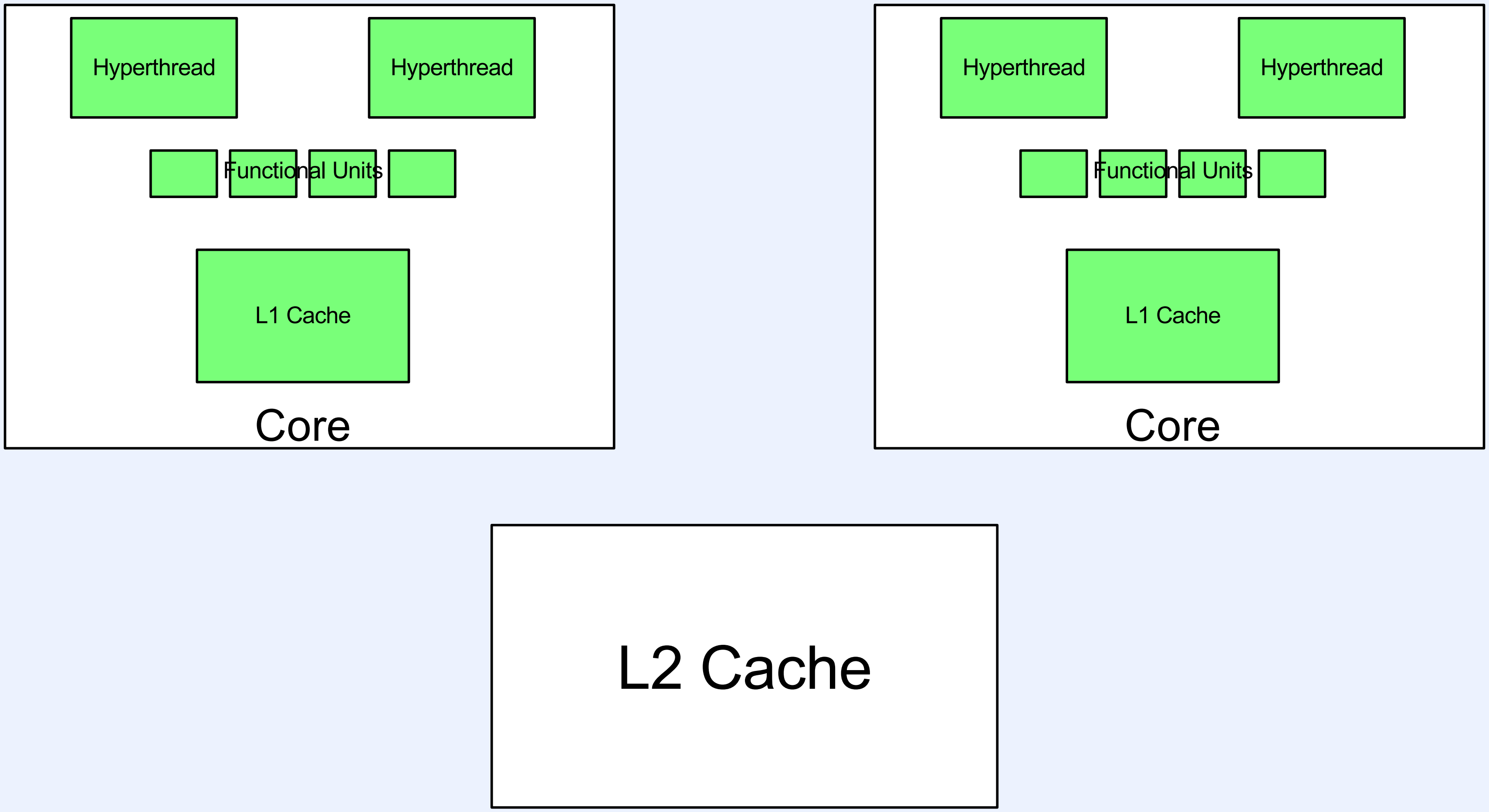
Quiz 1

We have a two-core processor with two hyperthreads per core. We have exactly two runnable threads.

a) **for best performance, each thread should run on a separate core**

b) **it doesn't matter which hyperthreads are used to run the two threads, as long as two hyperthreads are used**

Cores


Hyperthread
Hyperthread
Functional Units
L1 Cache
Core
Hyperthread
Hyperthread
Functional Units
L1 Cache
Core
L2 Cache

Quiz 2

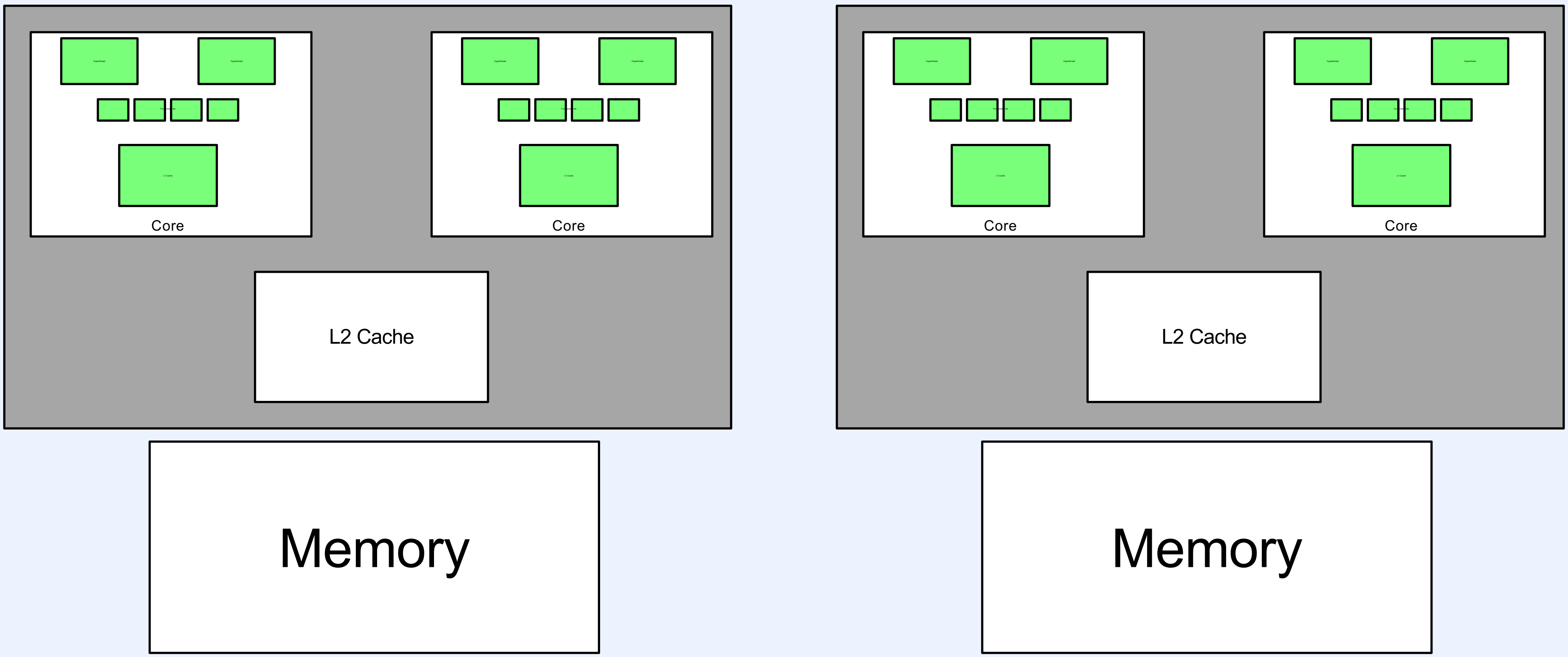
We have a two-core processor with four hyperthreads per core. We have four runnable threads.

a) it doesn’t matter which hyperthreads are used to run the four threads, as long as four hyperthreads are used

b) it does matter: we should make sure that two hyperthreads of each core are used

c) it does matter: not only should we make sure that two hyperthreads of each core are used, but we also need to consider whether any of the threads are in the same process

NUMA Nodes



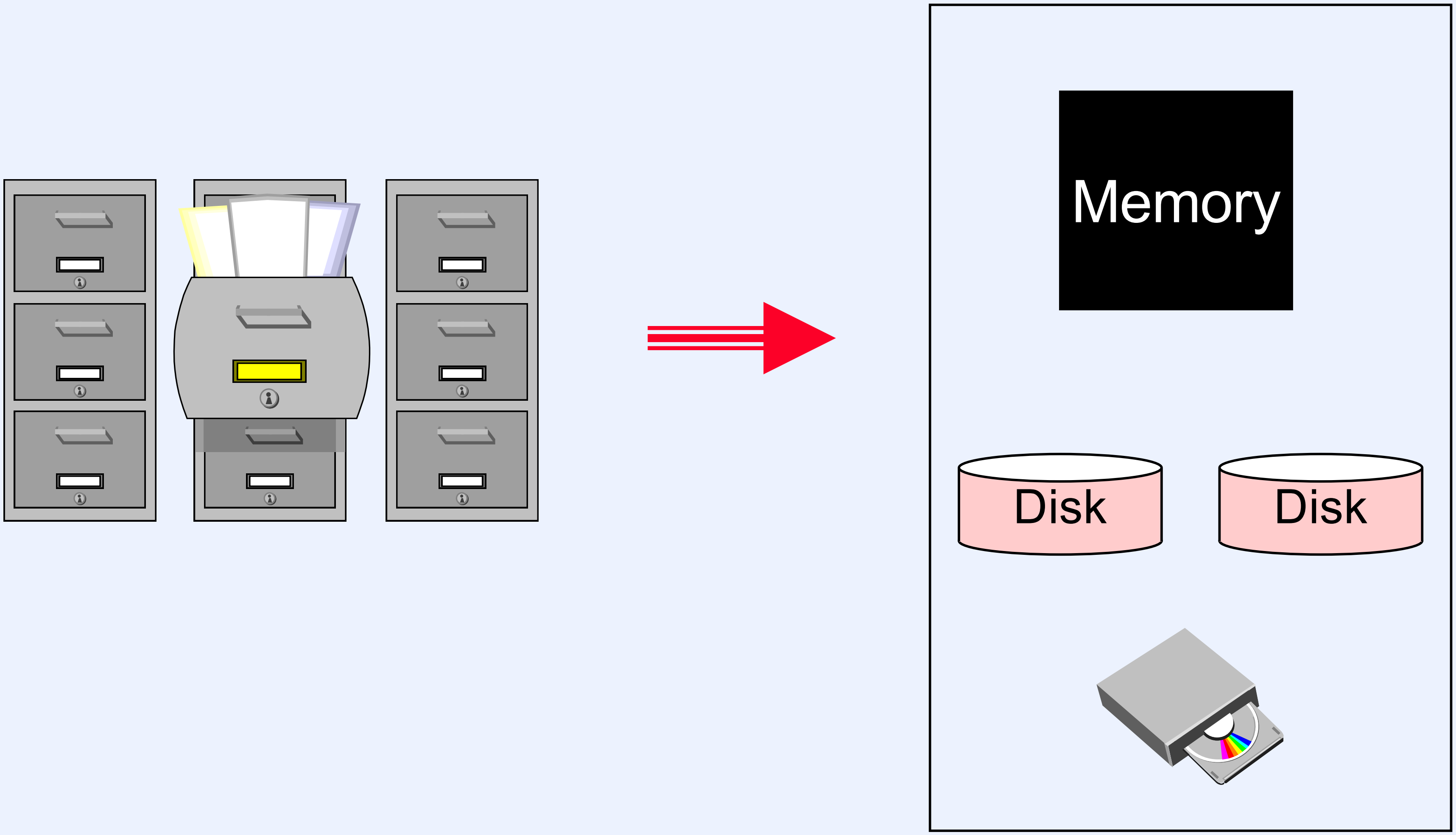
Quiz 3

We have a system with two NUMA nodes. Most of the hyperthreads are busy on one of the nodes, the other node is completely idle. What operation performed by a thread would make the thread (or the thread it creates) a good candidate to move to the other node?

a) pthread_create
b) fork
c) execv
d) waitpid

File Systems Part 1

Files



Requirements

- Permanent storage
  - resides on disk (or alternatives)
  - survives software and hardware crashes
    - (including loss of disk?)
- Quick, easy, and efficient
  - satisfies needs of most applications
    - how do applications use permanent storage?

Applications

- Software development
  - text editors
  - linkers and loaders
  - source-code control
- Document processing
  - editing
  - browsing
- Web stuff
  - serving
  - browsing
- Program execution
  - paging

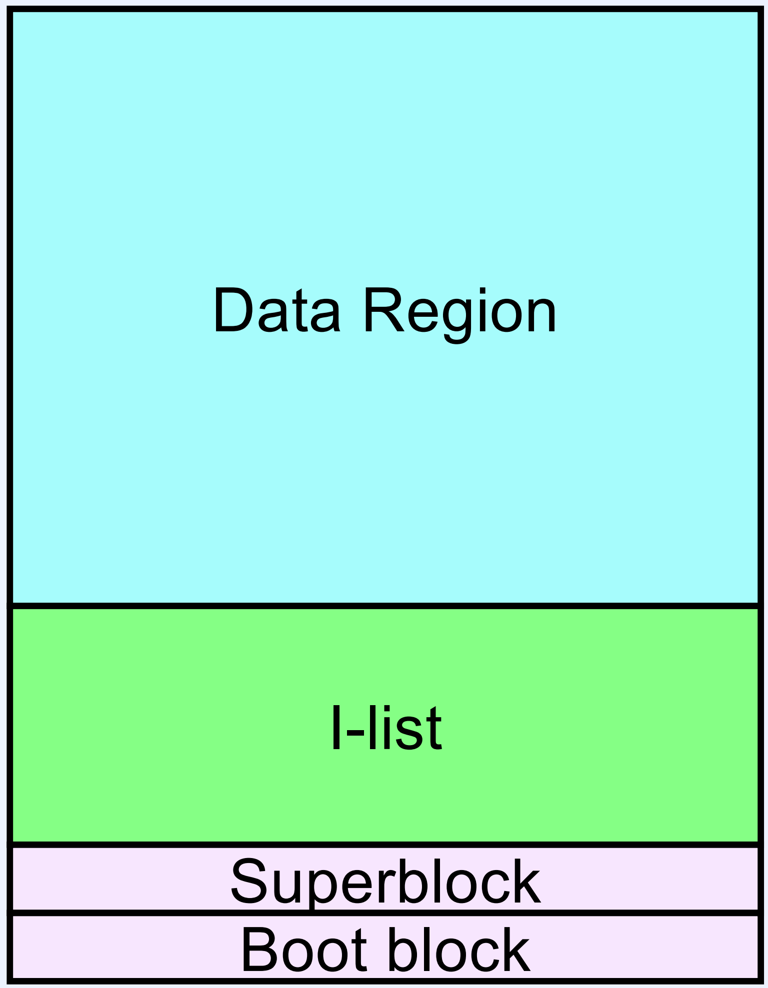
Needs

- Directories
  - convenient naming
  - fast lookup
- File access
  - sequential is very common!
  - “random access” is relatively rare

S5FS

- A simple file system
  - slow
  - not terribly tolerant to crashes
  - reasonably efficient in space
    - no compression
- Concerns
  - on-disk data structures
    - file representation
    - free space
- It’s the Weenix file system!

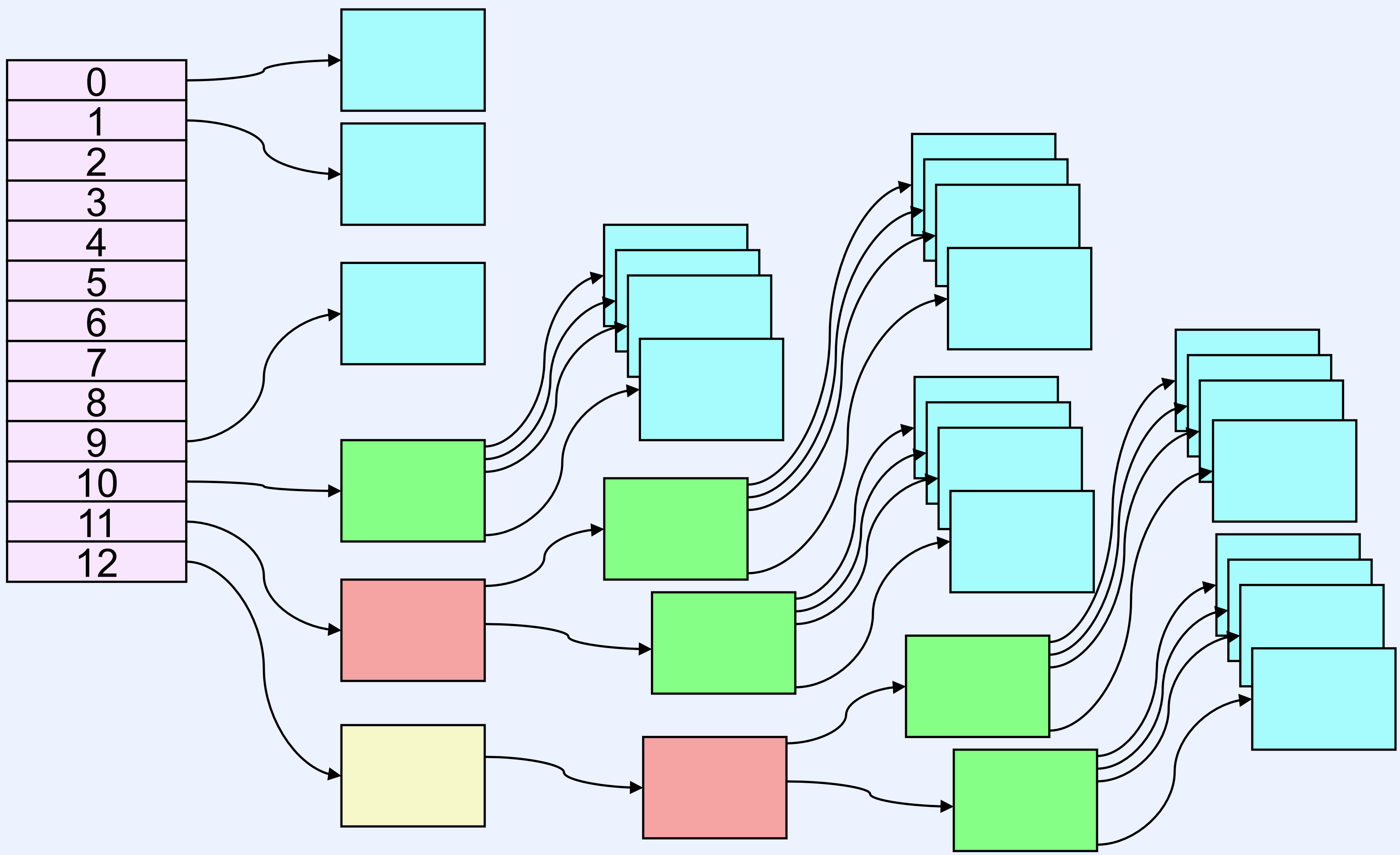
S5FS Layout



S5FS: Inode

Device
Inode Number
Mode
Link Count
Owner, Group
Size
Disk Map

Disk Map



Quiz 4

Suppose a new file is created. (At this point it occupies zero blocks.) Then one byte is written to it at byte offset $(266 \times 2^{10}) + 1$. Assume the block size is 2^{10} and block addresses occupy four bytes. How many blocks are required to represent the file, not counting its inode?

a) 1

b) 3

c) 270

d) more than 270

Quiz 5

Suppose one now writes to all locations in the file, from its beginning up to the location written to in the previous slide (byte offset $(266 \times 2^{10}) + 1$). How many blocks are required to represent the file, not counting its inode?

a) 1

b) 3

c) 270

d) more than 270