Concurrent Computing

Lecturers: Prof. Majid Mirmehdi (majid@cs.bris.ac.uk)
Dr. Tilo Burghardt (tilo@cs.bris.ac.uk)
Dr. Daniel Page (page@cs.bris.ac.uk)

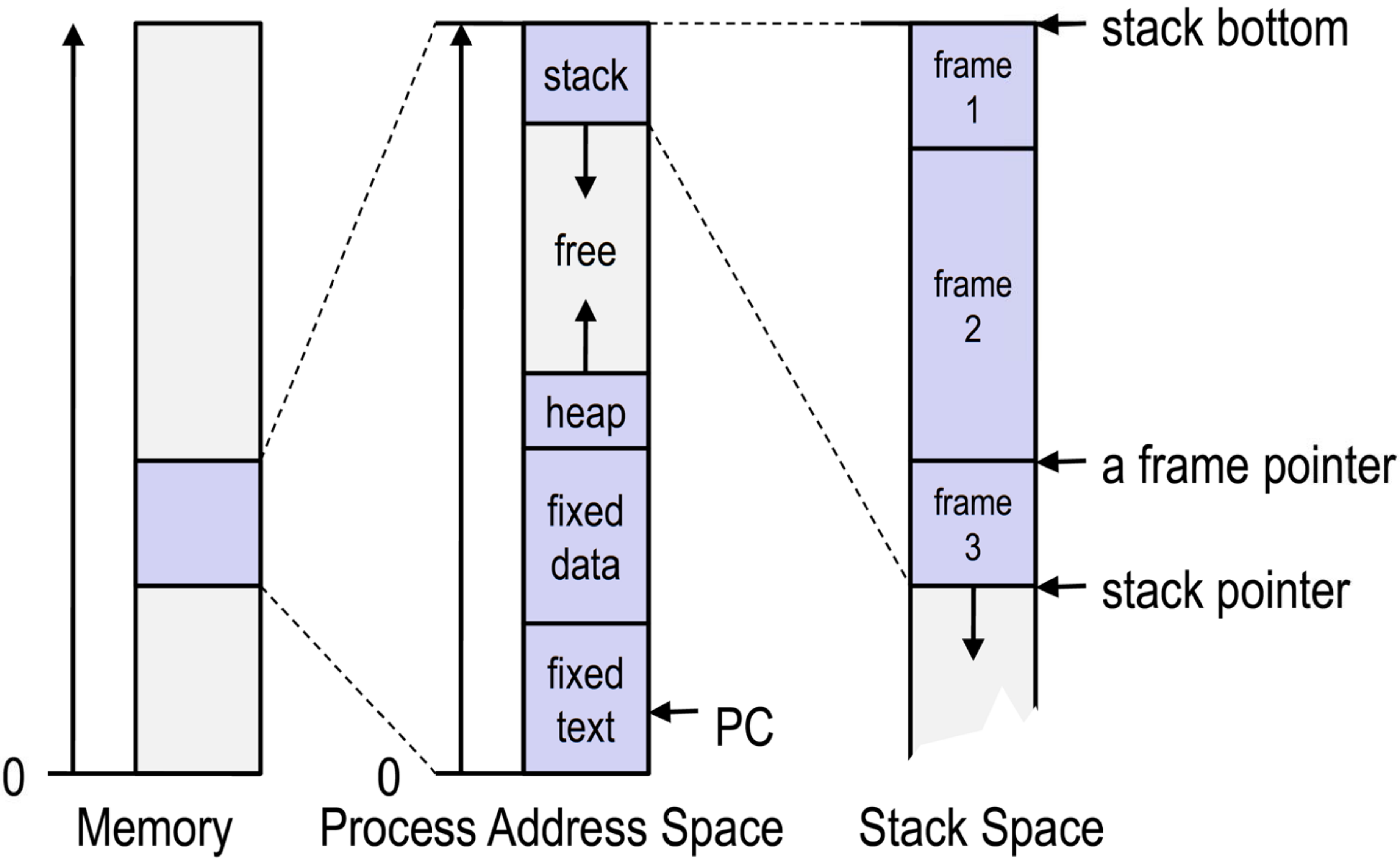
Web: http://www.cs.bris.ac.uk/Teaching/Resources/COMS20001

LECTURE 16

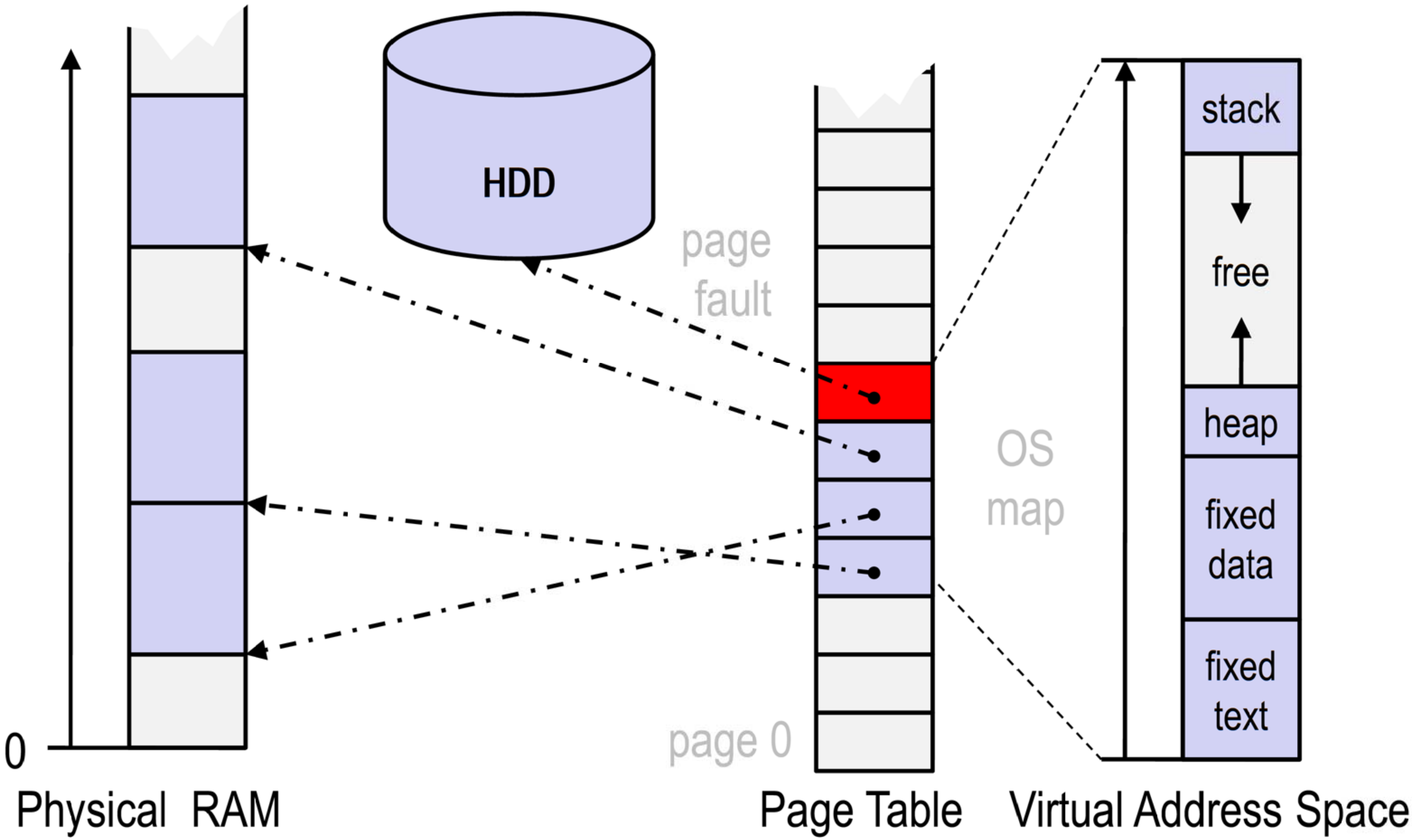
BASICS OF MEMORY & PROCESS ADMINISTRATION

Direct Physical Memory Access

(early PCs, some embedded systems etc)

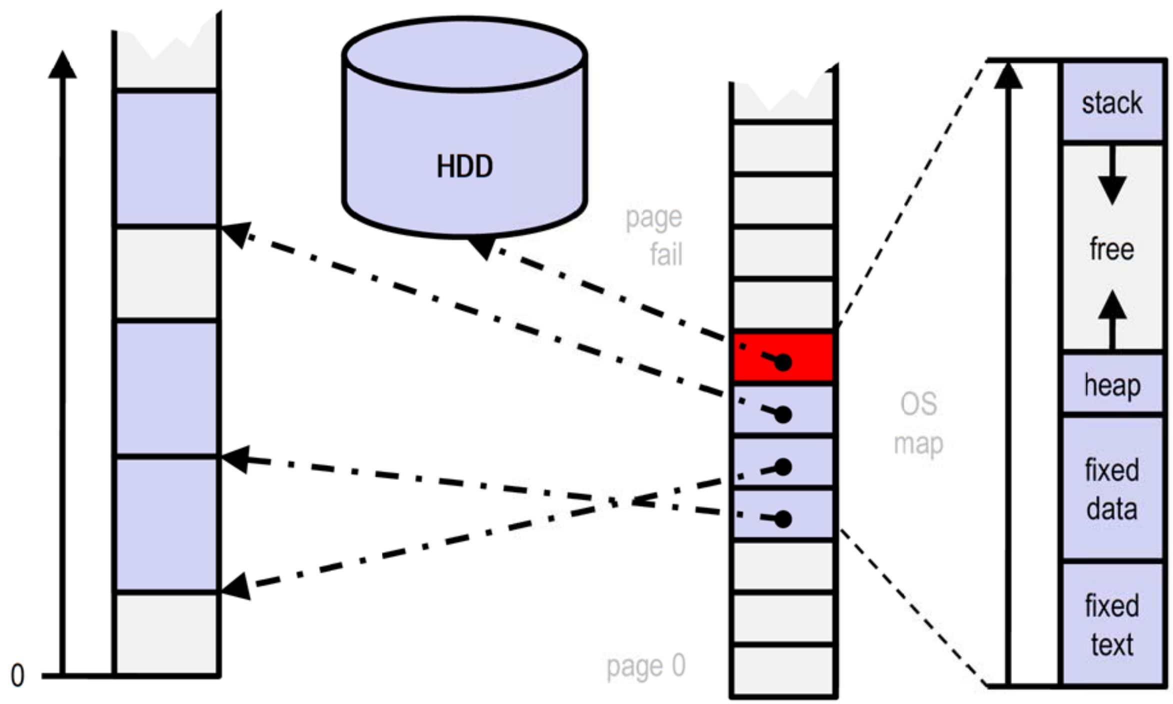


Idea of Virtual Memory (managed by OS)



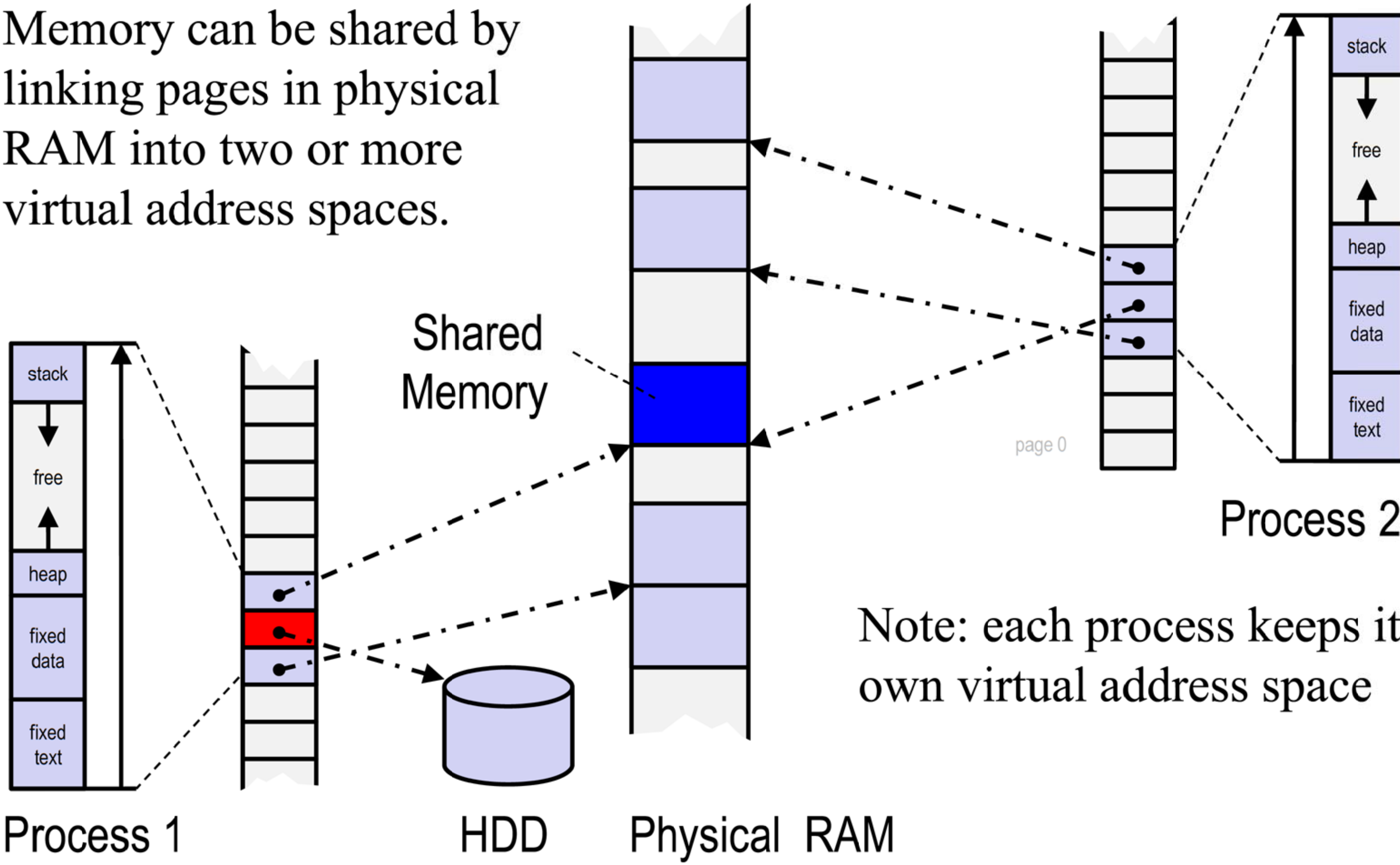
Aspects of Virtual Memory

- creates indirection (table look-up, entry is known as ***page***)
- creates illusion of single, large memory space
- requires twice the number of memory accesses → hold table in HW
- managed by operating system (OS)



Shared Memory via Virtual Memory

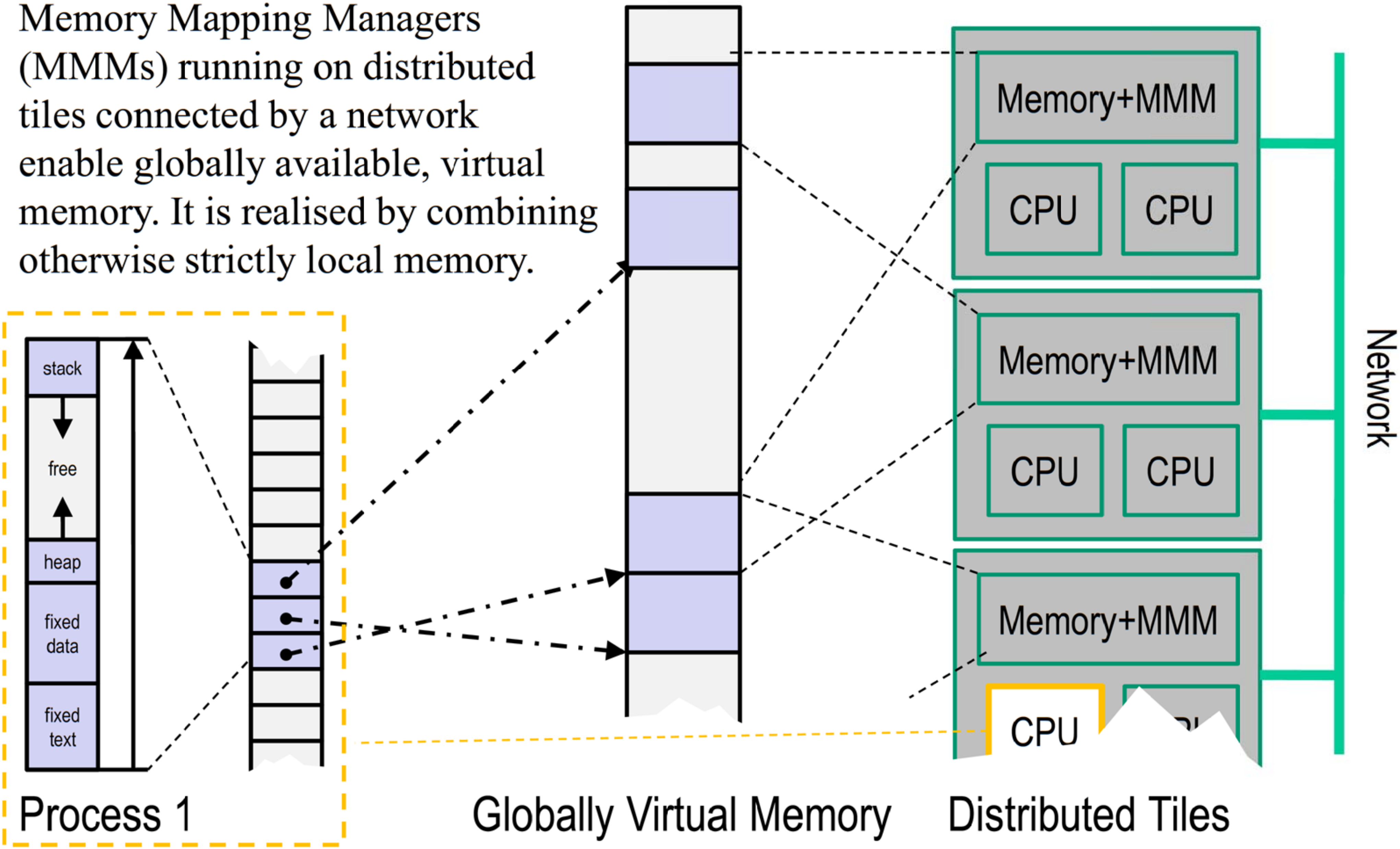
Memory can be shared by linking pages in physical RAM into two or more virtual address spaces.



Note: each process keeps its own virtual address space

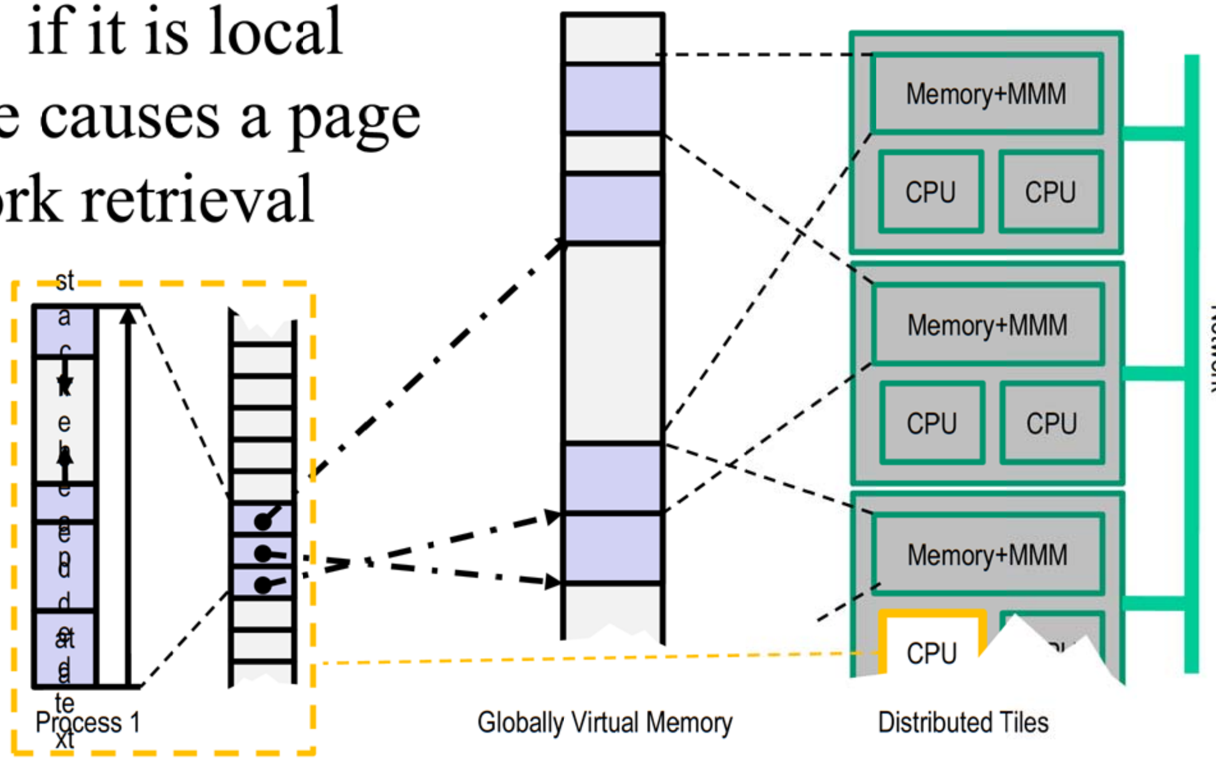
Distributed Shared Memory (via Globally Virtual Memory)

Memory Mapping Managers (MMMs) running on distributed tiles connected by a network enable globally available, virtual memory. It is realised by combining otherwise strictly local memory.

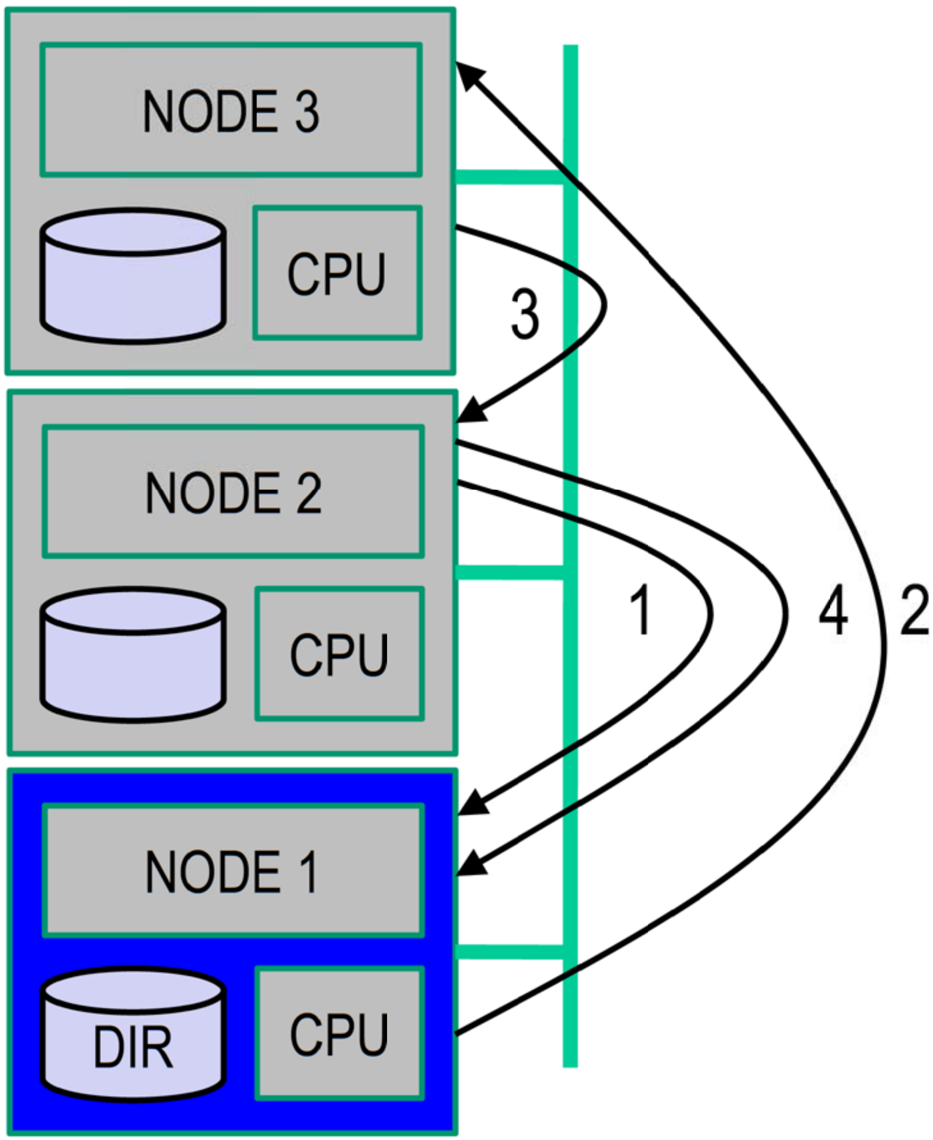


Concept of Distributed Shared Memory

1. Physical memory on each node holds pages of shared memory
2. Local pages are present in current node's memory whilst remote pages are in another node's memory and need to be fetched
3. Memory Management Unit (MMU) hardware often used to localise pages
4. Page table entry is valid if it is local
5. Access to non-local page causes a page fault and required network retrieval
6. DSM protocol handles page fault
7. Operations are transparent to programmer



Basic DSM Design: Central Directory Server



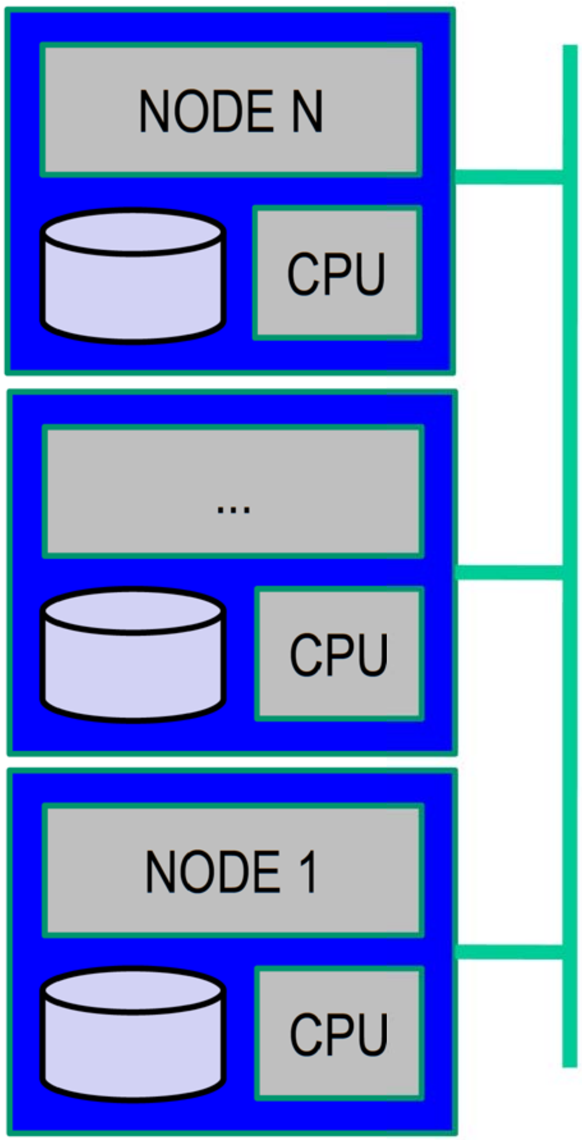
Key Design Idea:

- central server maintains directory –records hold which machine currently holds which pages

Page Localisation:

1. Node 2 page faults
2. Consult **central directory server** to locate page
3. Page requested from current owner, Node 3
4. Owner invalidates, sends to new location, Node 2
5. Node 2 informs directory of new ownership

Performance Issues of Central Directory Server



Problem 1:
directory at central server is a bottleneck

Improvement 1:
each node holds predictable subset of memory (e.g. Implemented via ***hashing*** of address space)

Problem 2:
each virtual page exists on only one machine

Improvement 2:
introduce ***caching*** to make read operations cheaper & parallel, yet write operations now become more expensive due to multiple updates

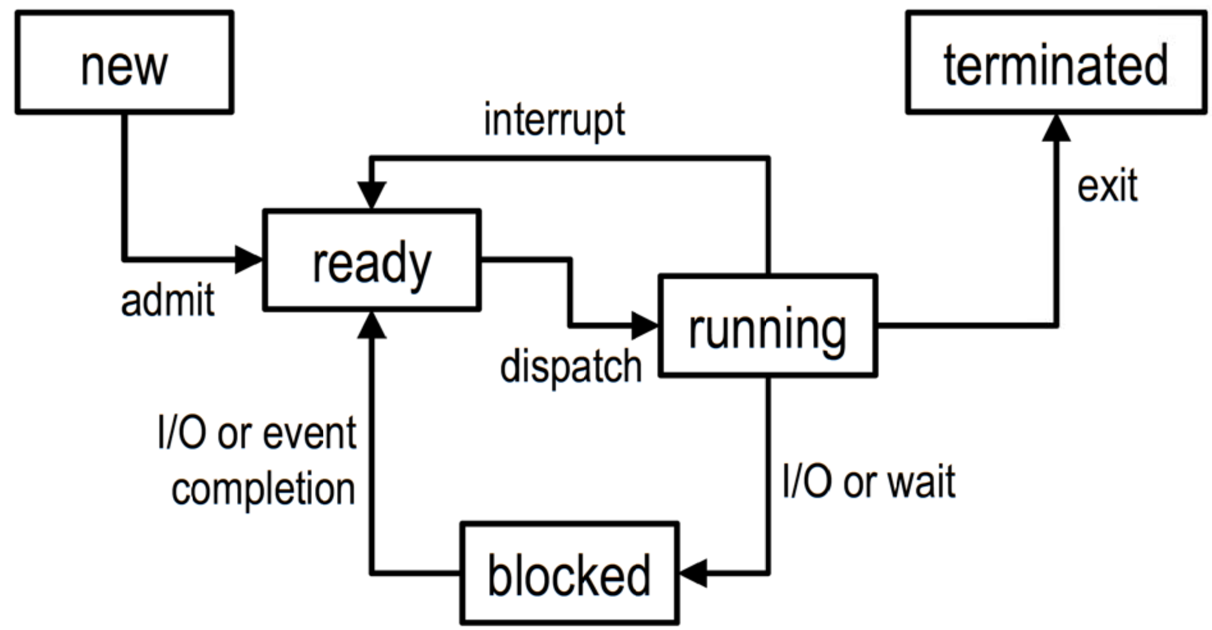
Concept of the Process Control Block (PCB)

The Process Control Block…

… is a data structure in the operating system that represents a process.

It may include:

- process id and state
- program counter
- registers
- heap memory info
- stack info
- list of open files
- inter-process info



Example Flow Chart of Process States

The Process Table…

…contains a PCB for each of the current processes in the system

Processes vs Threads I (with OS support)

- Process-based multitasking is running two or more programs ("processes") concurrently
- Thread-based multitasking is having a single program perform two tasks at the same time
- Whenever threads are used, the mother process is still used as a container for the state (the heap contents, and also OS resources such as priority, user ID, current directory, open file handles, network sockets, etc.) that all the threads in the process share.
- Every process contains at least one thread.
- Each thread contains its own stack, used to hold the method-local variables.
- Threads may also contain private (per-thread) "global" objects and state variables.

Processes vs Threads II (with OS support)

Threads of one process all share the same address space, in particular the same heap, yet own different stacks.

→ Fast switching + easy information exchange

→ No sharing of variables

Have you ever written a multi-threaded program in Java?

→ *Yes, all Java programs are multithreaded. The programs we've written so far run in a thread called main, the garbage collector runs in a different thread*

Why doesn't a Java program end when main returns?

→ *A Java program (the JVM process) ends only when all (user) threads have terminated.*