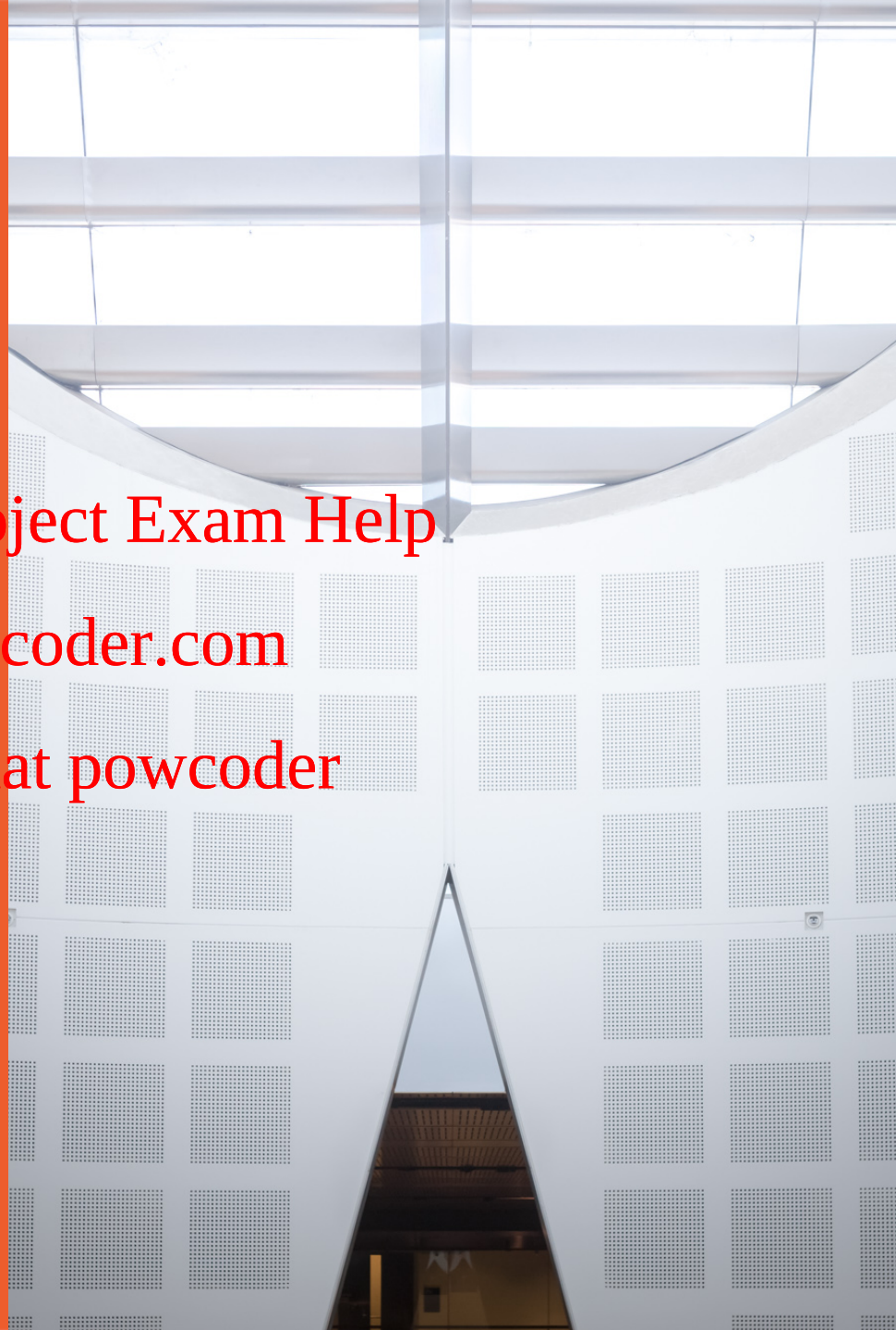
COMP3221: Distributed Systems

Architectures & Processes

Dr Nguyen Tran
School of Computer Science

THE UNIVERSITY OF SYDNEY



Previously…

- Basic definition of a distributed system
  - ***"A collection of independent computers that appears to its users as a single coherent system."***

- Real-world examples for distributed systems

- **Transparency** helps the users observe a single coherent system

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource is replicated
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource

Outline - Architectures

- Software Architectures

- System Architectures
  - **Centralized architectures**
    - The Client-Server Model
    - The Layered Organization
  - **Decentralized architectures**
    - The Peer-to-Peer Organization
  - **Hybrid architectures**
    - Edge server systems
    - Collaborative distributed systems

Software Architectures

Architectures & Processes
Week 2, COMP3221

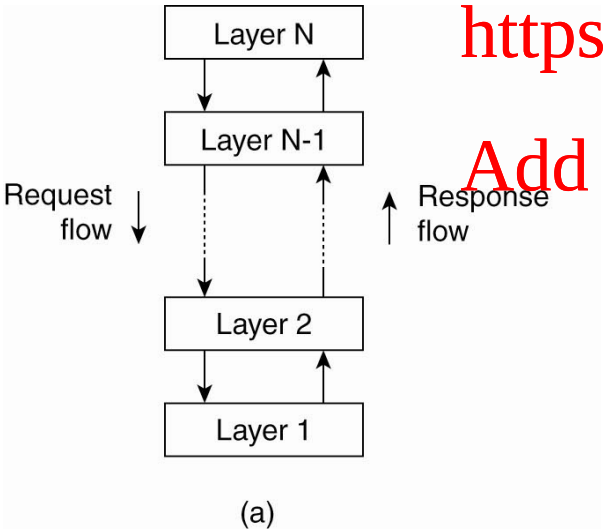
THE UNIVERSITY OF SYDNEY

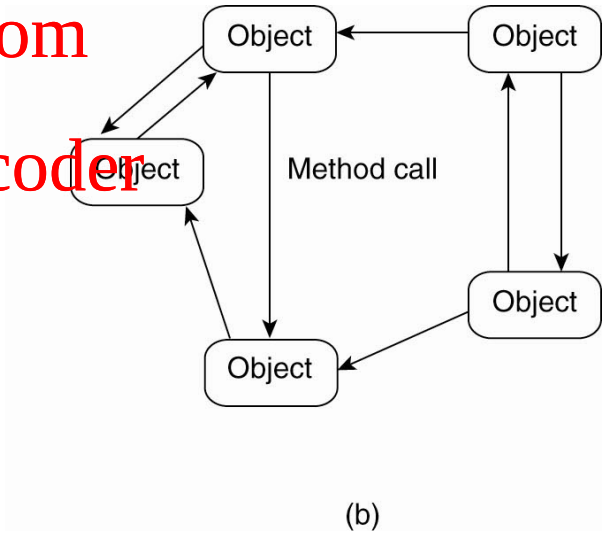
Software Architectures

Logical organization of components in distributed systems

- *Component:* A modular unit with well-defined interfaces that is replaceable
- *Connector:* a mechanism that mediates communication, coordination among components.

Layer-based Architectures vs. **Object-based Architectures**

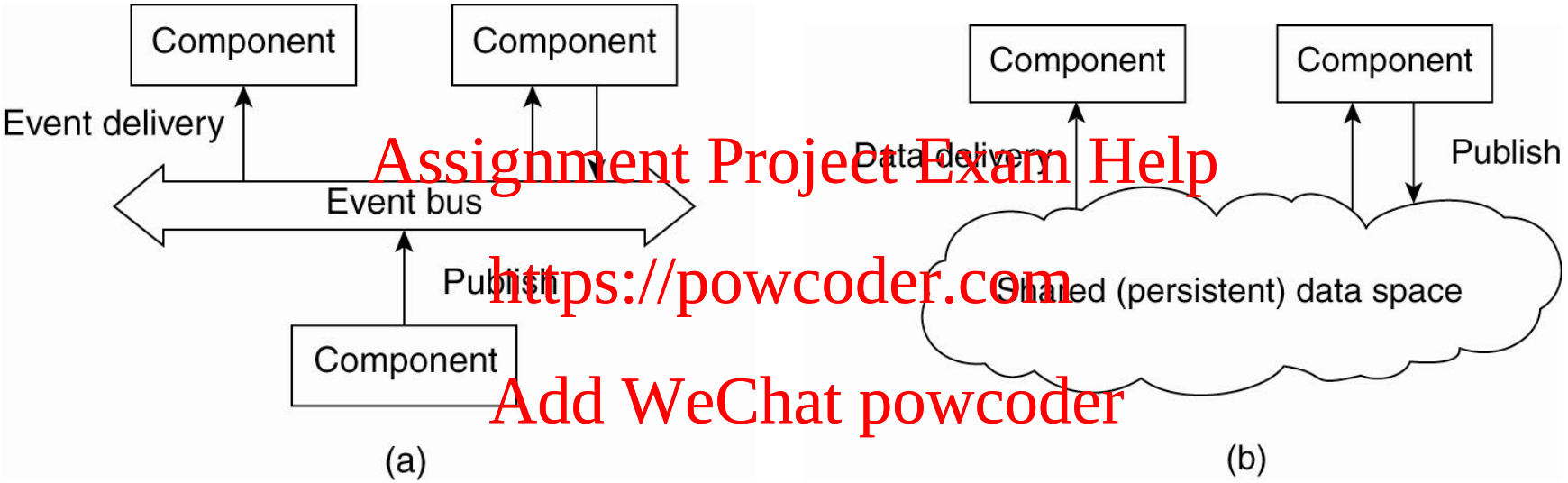




- In (a), requests (resp. responses) go downward (resp. upward)
- In (b), objects communicate through Remote Procedure Calls (RPCs)

Software Architectures

Event-based Architectures vs. **Data-centered Architectures**



- (a) Event-based architecture: communication through events, that optionally carry data (subscribers get their desired events delivered)
- (b) Data-centered architecture: through a shared repository, that contains data (e.g., files in a distributed file system, web-based distributed system, Twitter)

Centralized Architectures

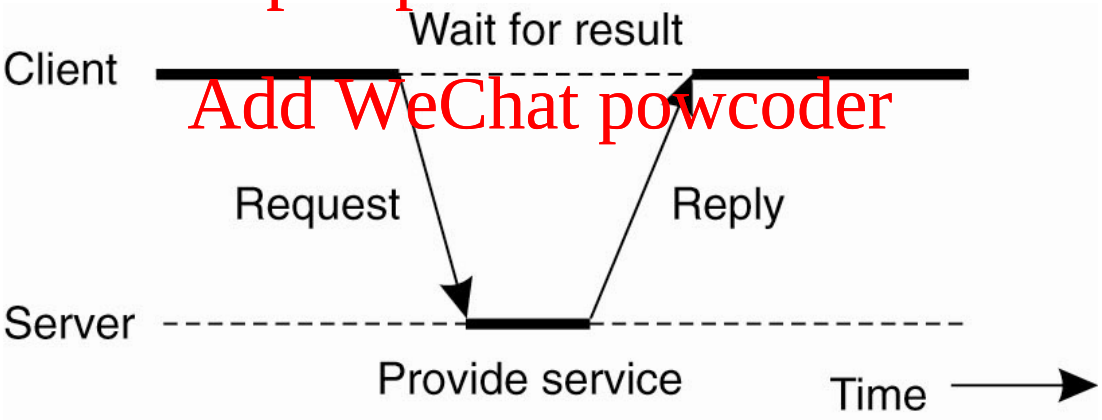
Architectures & Processes
Week 2, COMP3221

THE UNIVERSITY OF SYDNEY

Centralized Architectures

The basic client-server model

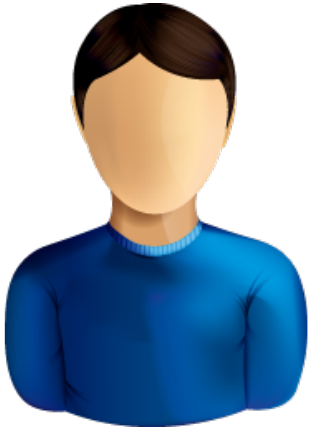
- The client/server requests/provides the services
- The client and the server can be hosted on different machines.



- The communication follows a request-reply model.
- Examples: ??

The Client-Server Model

Stateless vs Stateful server



Client

Hi, I'm comp1, may I have the lines 21-40 of file 5?

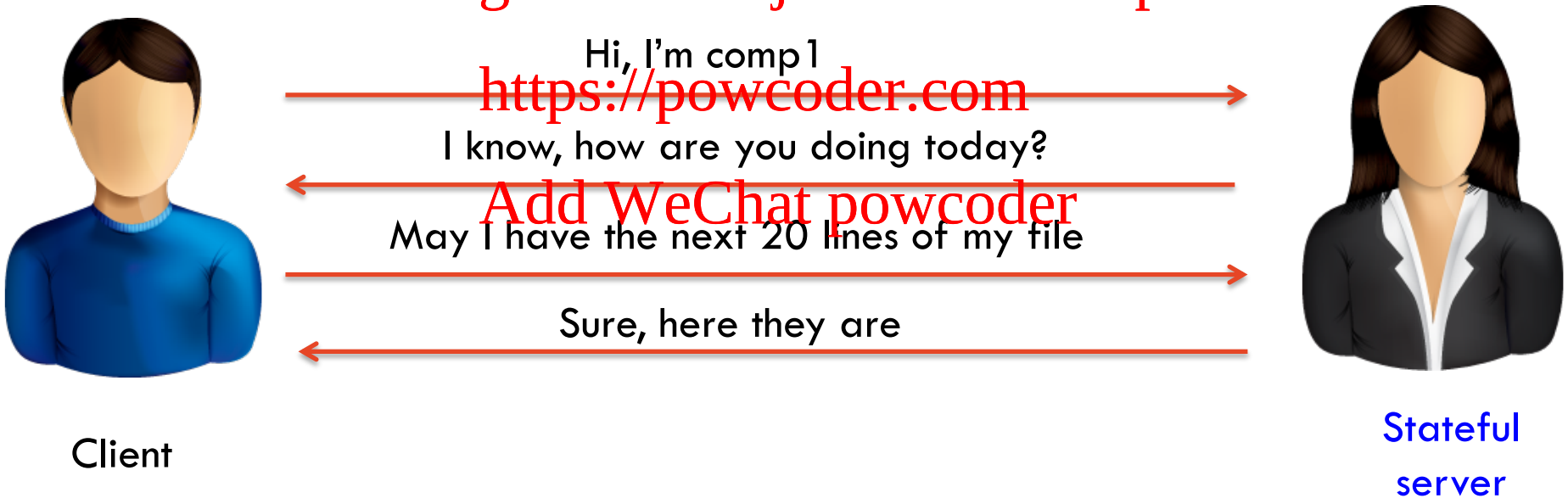
Sure, you have the credentials, attached are the lines

?

Stateless server

The Client-Server Model

Stateless vs Stateful server



The Client-Server Model

Stateless vs Stateful server

- Stateless server: does not record the state of its clients
- Stateful server: maintains persistent information about its clients (client->file)

	Stateless server	Stateful server
State	No info kept	Persistent info
Request	Self-contained	Can be split, generally faster
Upon failure	No recovery needed	State recovery needed (explicit deletion)
Example	Network file system (NFSv3)	Andrew file system (AFS)

The Layered Organization

Architectures & Processes
Week 2, COMP3221

THE UNIVERSITY OF SYDNEY

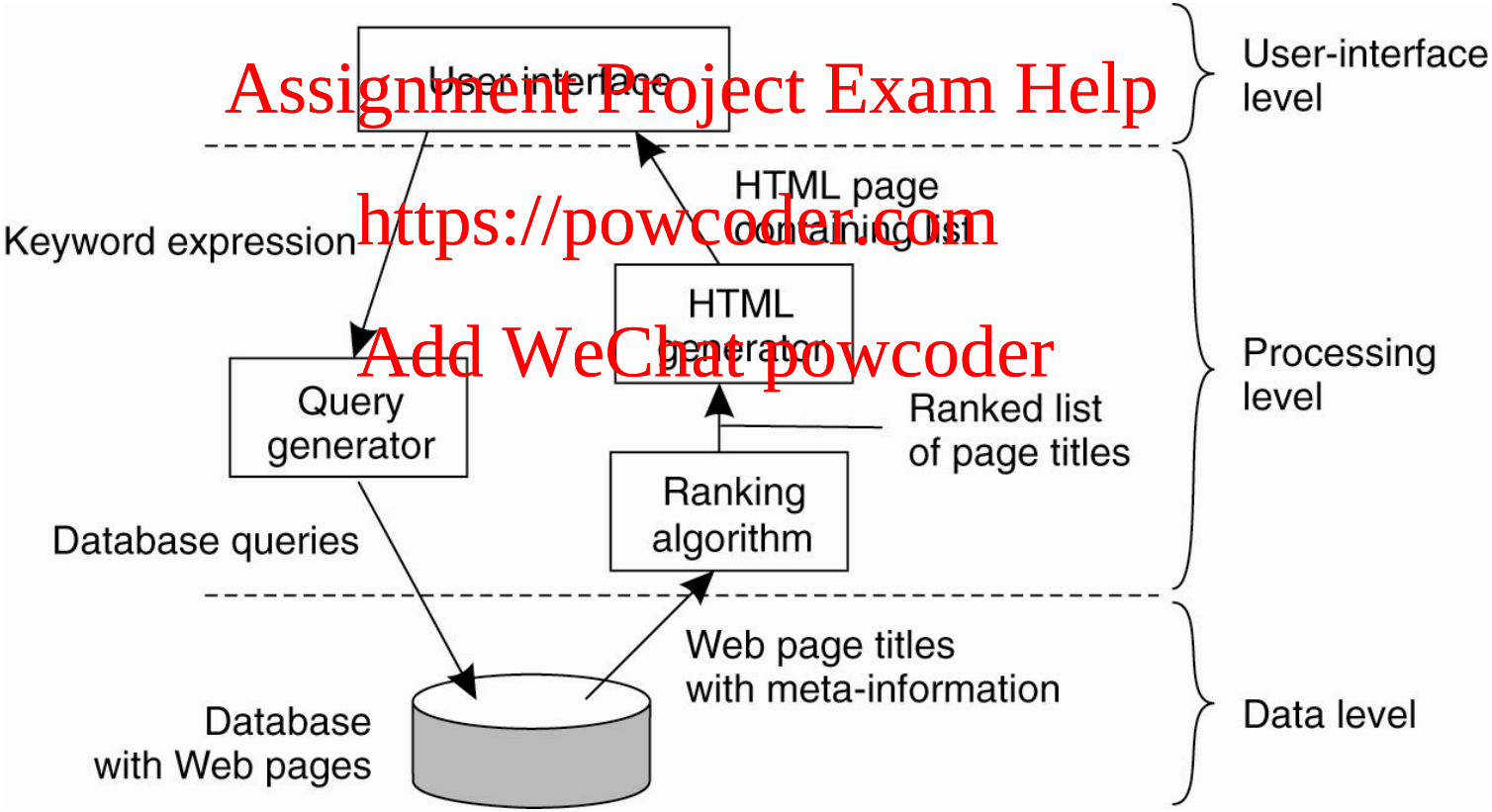
Application layering

Traditional three-layered view:

1. **The user interface layer**
   - contains the feature to control the application
2. **The processing layer**
   - contains the function of the application
3. **The data layer**
   - contains the data of the application

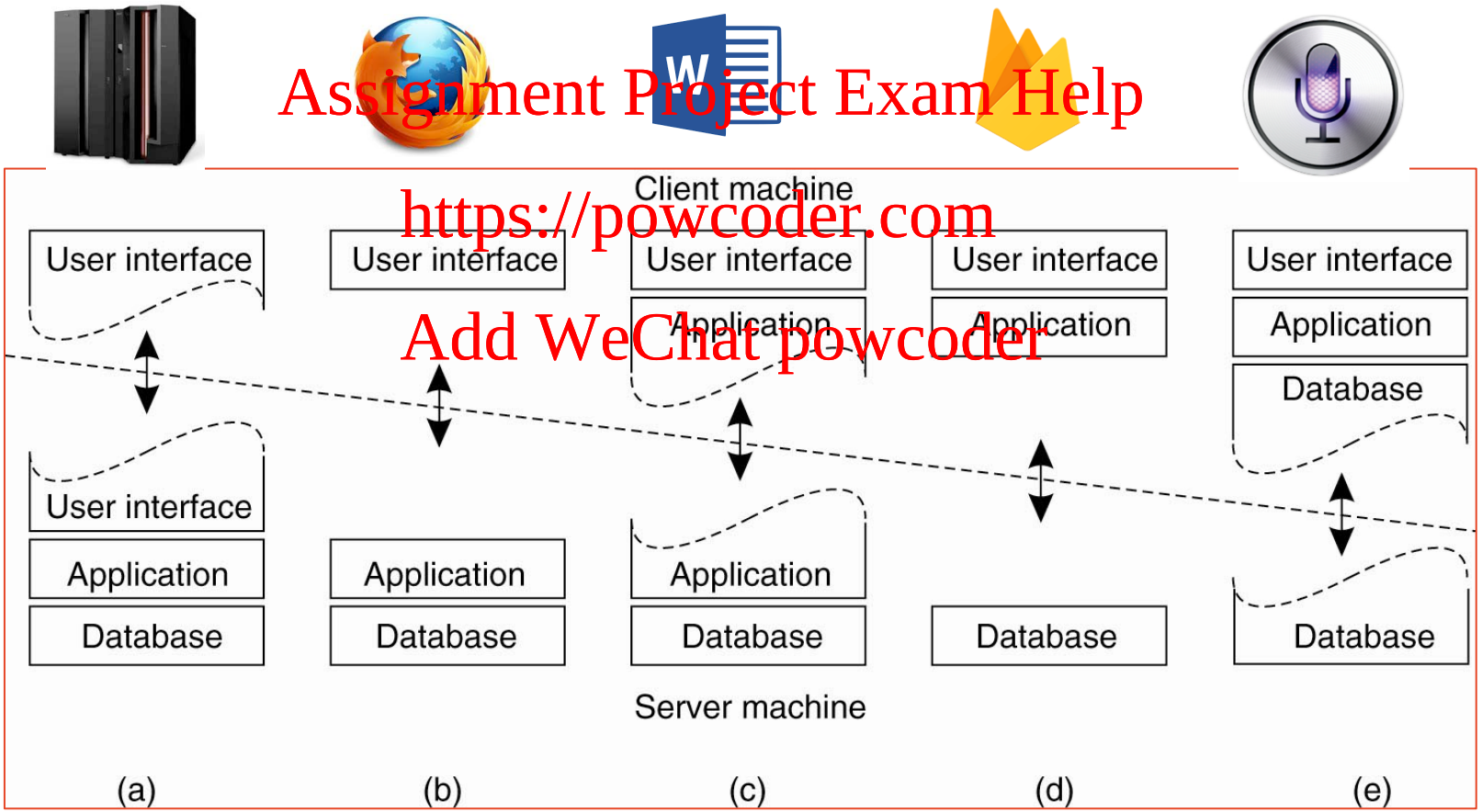
Application layering (cont'd)

Example: a search engine request spanning the traditional three layers



Multi-tiered architectures

Physical two-tiered architecture



Multi-tiered architectures (cont'd)

A single machine can act both as a client and a server

Example: Cloud computing

- *Cloud computing*: the delivery of computation or storage as a service to end-users.
- The servers handle most of the computation upon request and sends back the results to the client
  - One server asks data to another server
  - Another does the computation
  - …

Decentralized Architectures

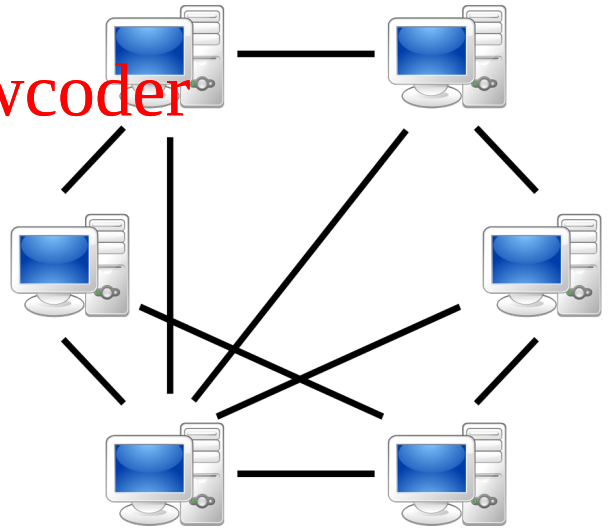
Architectures & Processes
Week 2, COMP3221

THE UNIVERSITY OF SYDNEY

The peer-to-peer model

Every machine acts similarly

- Every machine is both a client and a server
- No centralized control: the responsibility is distributed evenly
- Even the program executing on each machine is similar
- Forms an **overlay network,**
  - nodes are processors,
  - links possible communication channels.



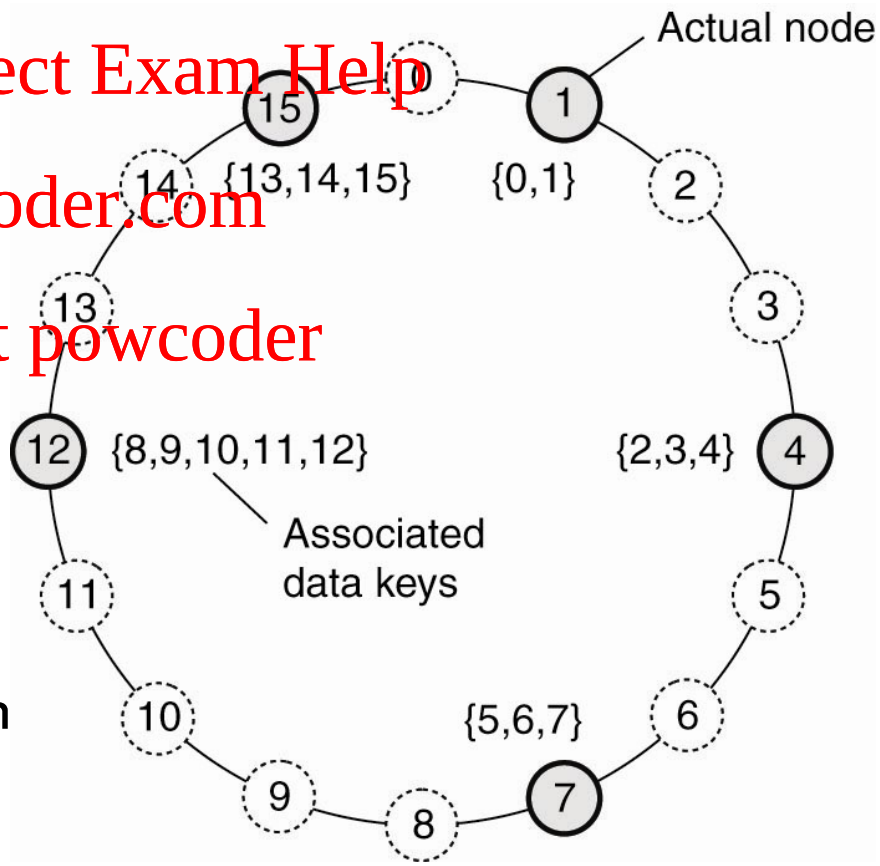
Structured Peer-to-Peer Model

The overlay network is constructed using a deterministic procedure

- **Chord** is an example of a **Distributed Hash Table (DHT)**

As a node:

- I have a successor peer
- I have a predecessor peer
- I have some shortcuts to other nodes to speedup delivery of requests
- I am responsible of a subset of the system data items (based on my unique identifier)



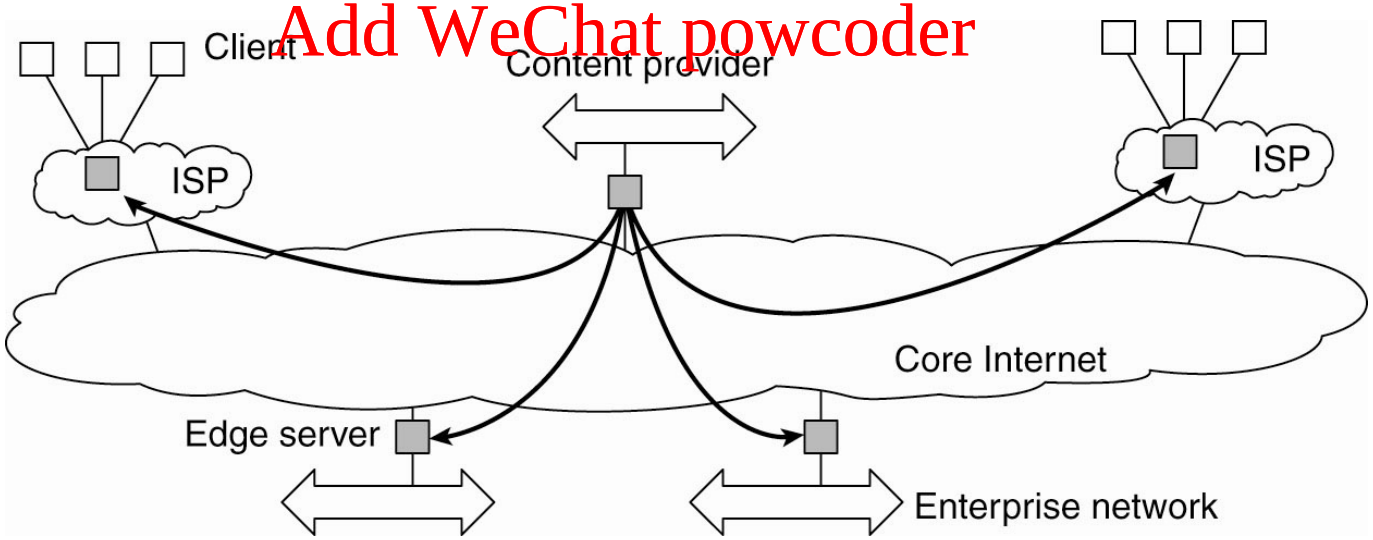
Hybrid Architectures

Architectures & Processes
Week 2, COMP3221

THE UNIVERSITY OF SYDNEY

Edge-Server Systems

- Servers are placed at the edge of the network.
- Edge servers serve content, possibly after applying filtering and transcoding functions
- One edge server acts as an origin server for all content, while taking advantage of other servers for optimized delivery of content, e.g. through replication.



Collaborative Distributed Systems

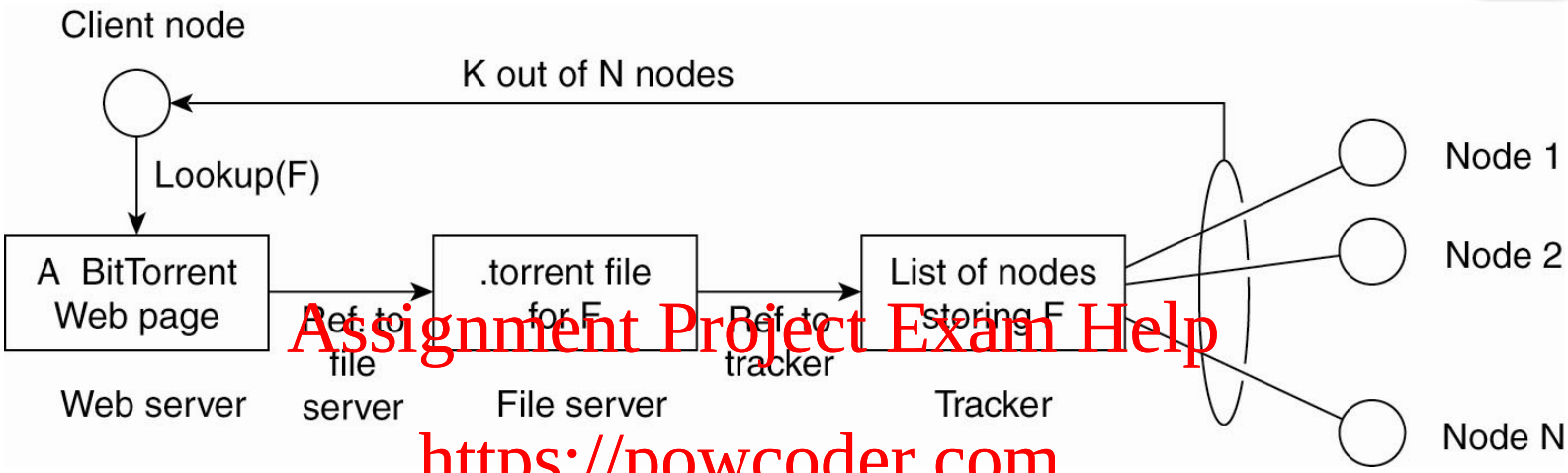
- How do you first get started ?
  - Often a traditional client-server models is used.
  - Once joined fully decentralized schemes can be used.

- Example: **BitTorrent**, a file sharing application

- 35% of world-wide internet traffic in 2015.

- Used for Linux distribution, software patches, distributing movies
- Goal: quickly replicate large files to large number of clients

BitTorrent



- Web server hosts a .torrent file (w/ file length, hash, tracker's URL…)

- A tracker (server or a DHT) tracks downloaders/owners of a file
- Files are divided into chunks (256kb-1MB)
- Downloaders download chunks from themselves (and owners)
- **Tit-for-tat:** the more one shares (server), the faster it can download (client)

Conclusion

- Client and server are used to identify the role of communication participant

- Client and server roles may run on:
  - The same machine
  - Distinct machines with very different resources
  - Distinct machines with similar resources

- Most moderns distributed systems are hybrid architectures

Processes

Architecture & Processes
Week 2, COMP3221

Dr. Nguyen Tran
School of Computer Science

THE UNIVERSITY OF SYDNEY

Introduction

- **Previously:** a distributed system gives the illusion of a single system to a user thanks to transparency
- **Now:** a single system gives the illusion of multiple resources to multiple users thanks to concurrency

Outline - Processes

- A Very Brief History
- UNIX Processes and Threads
- Multi-threading
- Multi-threaded Server

A Very Brief History

Architectures & Processes
Week 2, COMP3221

THE UNIVERSITY OF SYDNEY

Evolution of Operating Systems (OSes)

- The hardware evolved
  - Hardware was expensive, humans were cheap (e.g., Multics 1960's)
  - Hardware was cheaper, humans were expensive (e.g., desktop computer 1980's)
  - Hardware are very cheap, humans are very expensive (e.g., handheld devices 2000's)
- The OS interaction had to adapt
  - Batch: one execution at a time
  - Multiprogramming: multiple program executions simultaneously
  - Timeshare: split time into slots allocated for different program execution
  - GUI: multiple interfaces to access a system from different end-points
  - Ubiquitous devices: each user possesses her own computational device

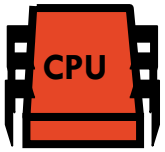
Uniprogramming vs. Multiprogramming

- **Uniprogramming**: one program execution at a time
  - MS/DOS
  - early Macintosh
  - Batch processing
  - No longer acceptable!

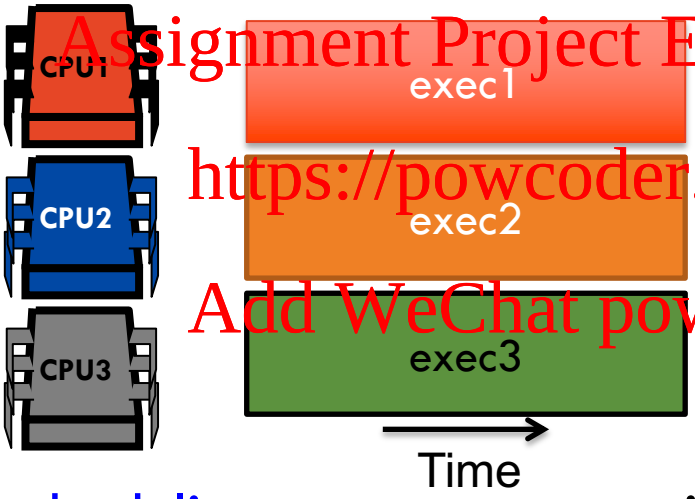
- **Multiprogramming**: more than one program execution at a time
  - Multics, UNIX/Linux, Windows NT, Mac OS X

- Many program executions on personal computers:
  - Browsing the web
  - Sending emails
  - Typing a letter
  - Burning a DVD

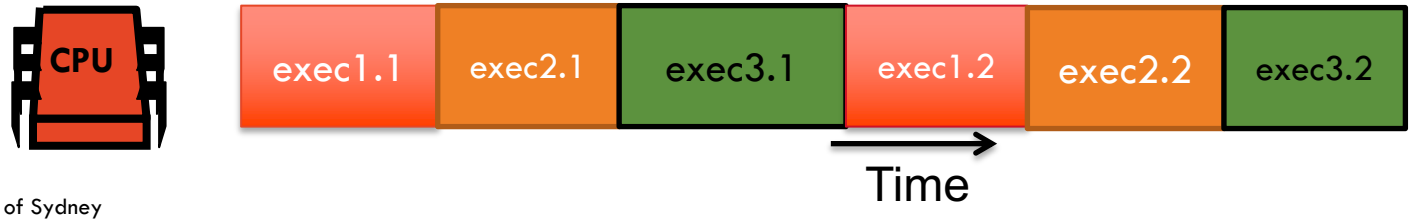
Concurrency



- Assume a single central processing unit (CPU)
- **Problem: How to give the illusion to the users of multiple CPUs?**

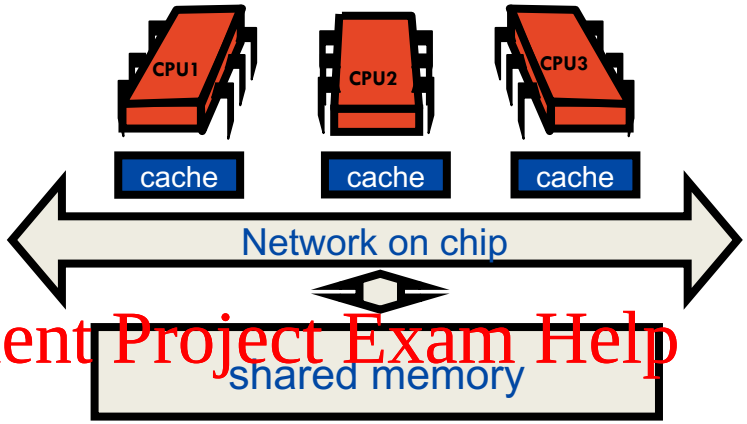


- Solution: scheduling program executions, one after the other during small fractions of the time

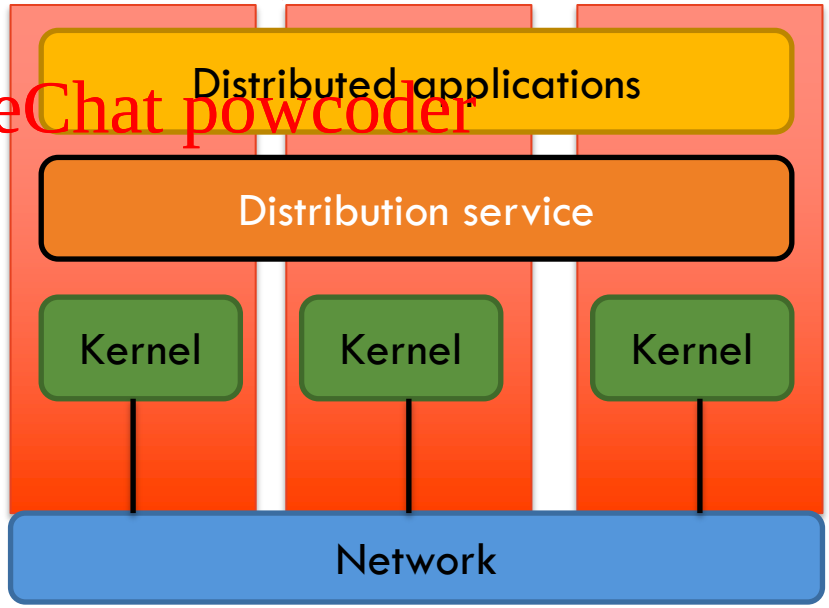


Concurrency

- Multi/many-cores on a single machine

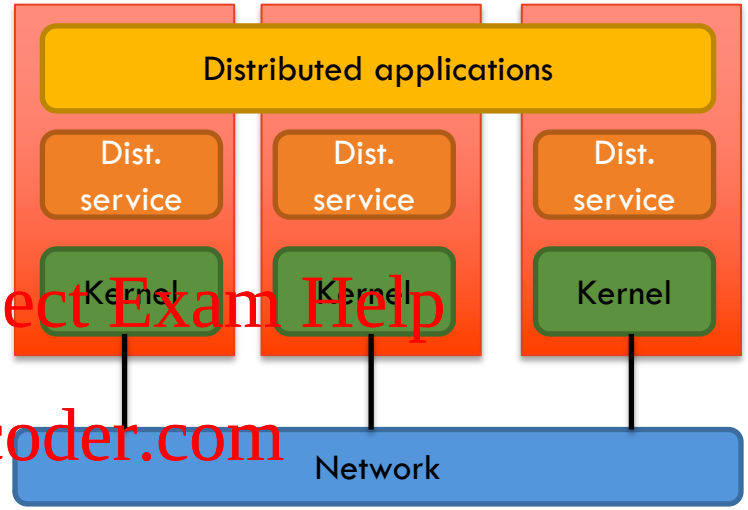


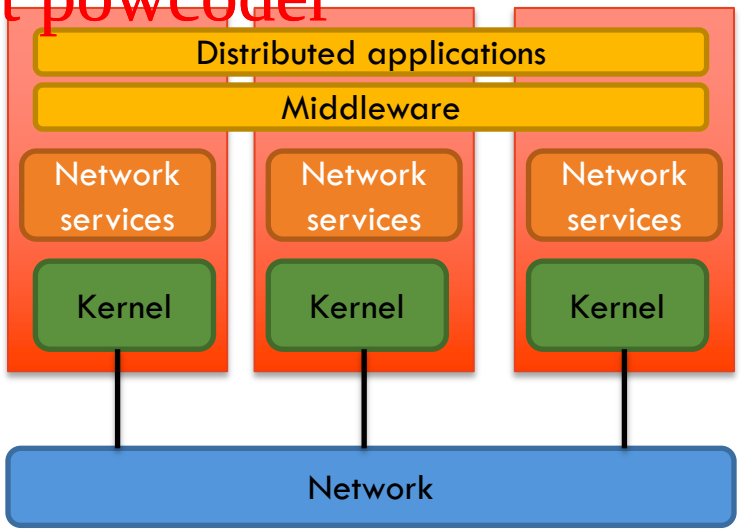
- Distributed operating systems
  - This is a single system image, the system maintains a single copy of the resources



Concurrency

- Network operating systems
  - Machines provide resources to other machines (e.g. UNIX rlogin)
- Middleware
  - A layer over the network services providing general services to applications in a very transparent manner (systems can differ)





Processes and Threads

Architectures & Processes
Week 2, COMP3221

THE UNIVERSITY OF SYDNEY

Processes

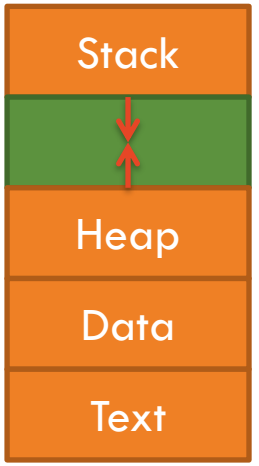
Process: defined as a program in execution

- Operating system creates a virtual CPU for each process
- OS makes sure that processes cannot maliciously or inadvertently affect each other's behaviour, i.e. Secure !
- This concurrent transparency comes at a very high cost:
  - For every process (virtual CPU), OS must create a completely independent address space.

Address space

***Address space*: A unit of management of a process's virtual memory**

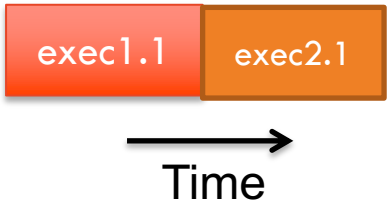
- Process address space:
  - Stack: temporary data extensible towards lower virtual addresses
  - Heap: memory allocated dynamically extensible to higher virtual addresses
  - Data section: global variable
  - Text region: program code

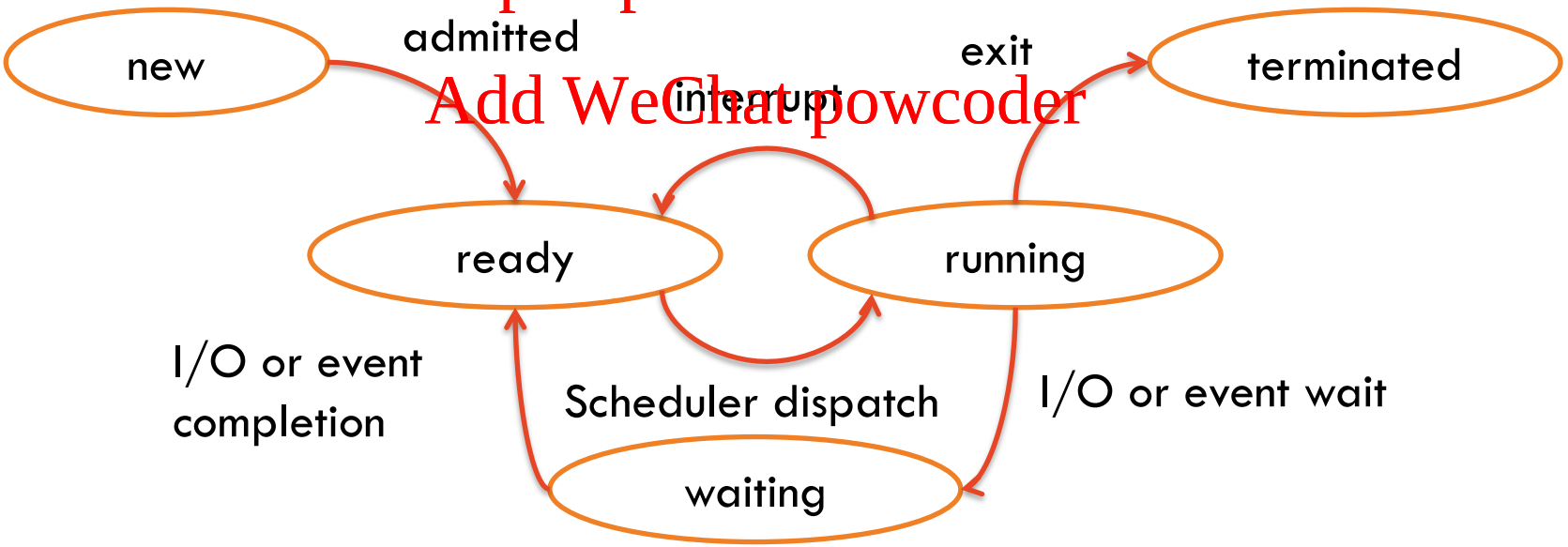


Process

***Process*: an abstraction representing a program execution**

- A process switches from a state to another:
  - When there is timer interrupt goes off, explicit yield, I/O, etc.





Process management

Process creation:

- Process allocation: the act of choosing the host of the process
  - The system provides command to execute a process at an idle workstation
  - The system chooses a host from a pool of processors to execute it
- Execution environment: an address space w/ initialized content & open files
  - Static: program text, heap and stack regions created from a list
  - Dynamic: UNIX fork shares program text and copies stack and heap regions

Process management

- Switching from one process to another has become one of the most expensive operations due to **Context-Switches**
  - CPU context: register values, program counter, stack pointer, etc.
  - Modify registers of MMU (Memory management unit)
  - Invalidate address translation caches such as in TLB (Translation lookaside buffer)

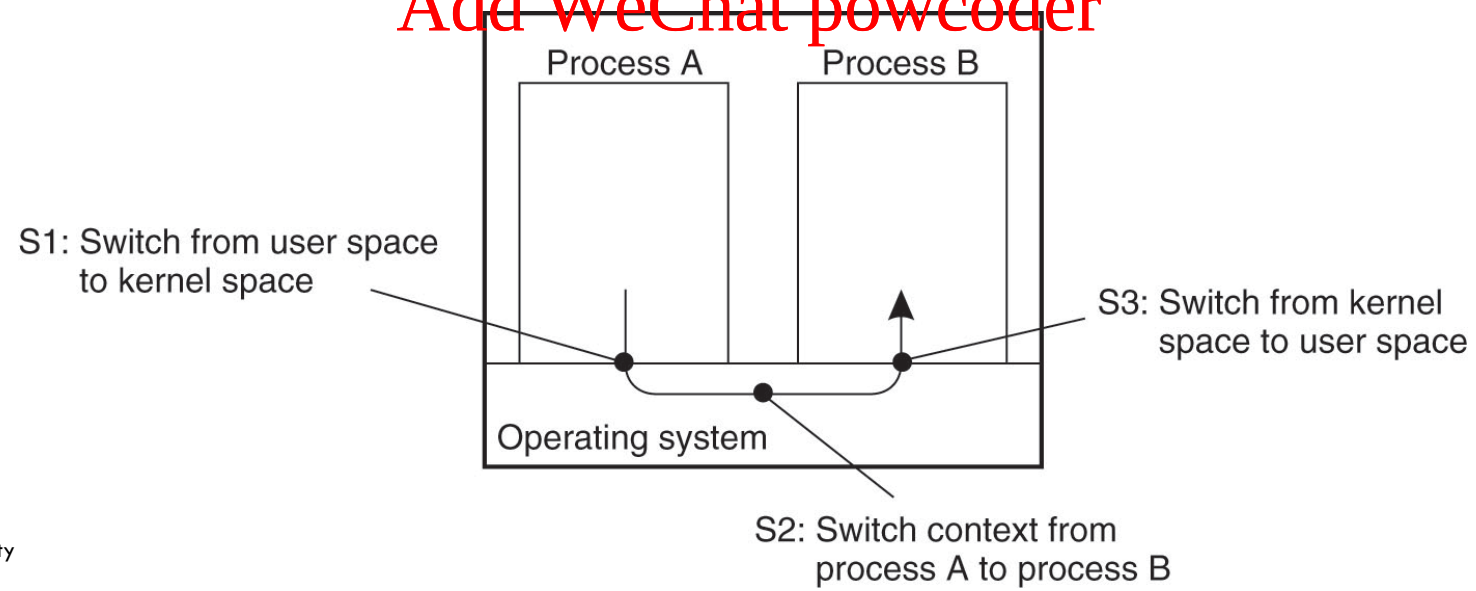
Inter-Process Communication (IPC):

- Pipes/Message queues/Shared memory segments
- Requires costly context switches

Context-switch

IPC requires kernel intervention

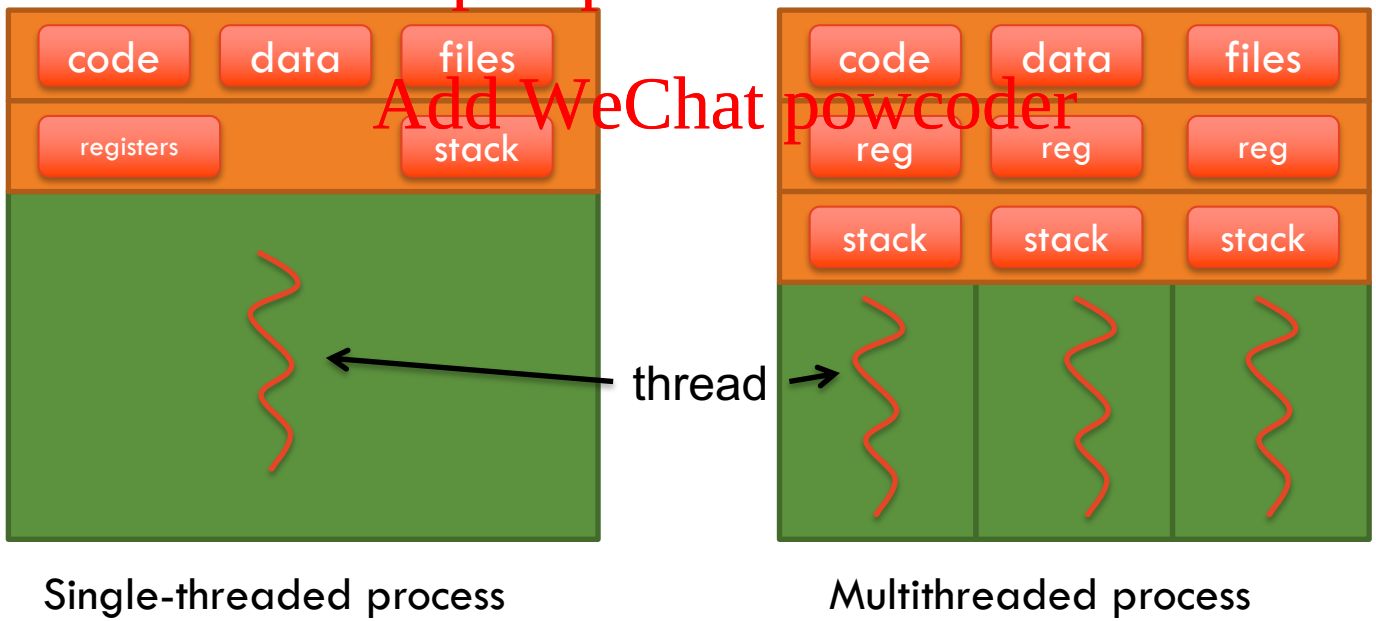
- S1. Process A switches from user to kernel mode
  - Changing the memory map in the MMU (memory management unit).
  - Flushing the TLB (translation look aside buffer)
- S2. Process context-switch (swapping processes from A to B)
- S3. Process B switches from kernel to user mode
  - Again requires changing the MMU map and flushing the TLB



Thread

***Threads*: smallest unit of CPU utilization**

- Multiple threads per process
- Portable OS Interface (POSIX) Threads
  - Current activity: program counter
  - Stack: temporary data
  - No data, code, files (but it shares them with other threads)



Single-threaded process Multithreaded process

Thread

- Portable OS Interface (POSIX) Threads
  - Current activity: program counter
  - Stack: temporary data
  - No data, code, files (but it shares them with other threads)

- Communication between threads always through memory
  - Does not need IPC
  - No context switches required (when purely at user-level)

Thread Implementation

User-level threads library vs. kernel-scheduled thread

- **User-level thread library:**
  1. Cheap to create and destroy
  2. Blocking system call freezes the entire process of the thread (and other threads)
  3. Cheap context-switch:
     - Few instructions to store and reload CPU register values
     - No need to change memory map in MMU
     - No need to flush the TLB

- **Kernel-scheduled threads:**
  1. Costly to create and destroy
  2. Do not block the current process (and other threads) upon blocking system calls (I/O)
  3. Costly context-switch (similar to process context switch)
     - Needs to change memory map in MMU
     - Needs to flush the TLB

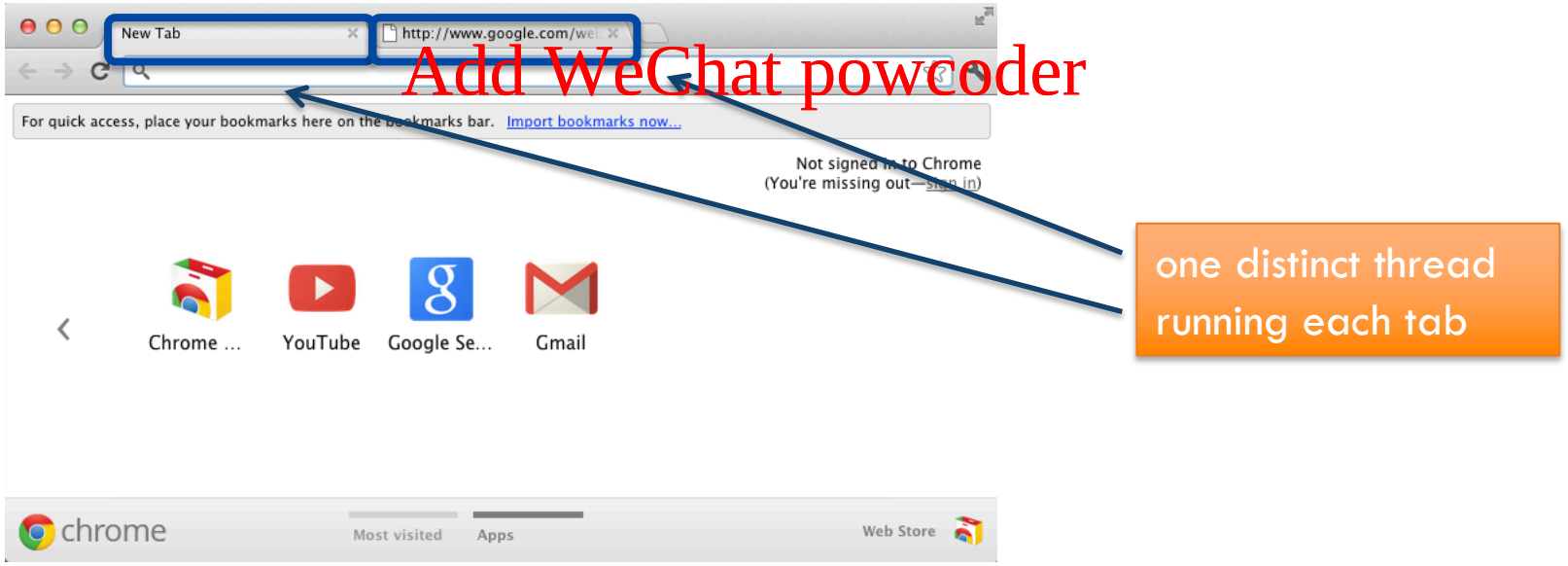
Processes vs. threads

- Processes
  - Isolated: prevents one process from interfering with another
  - Inefficient: starting/terminating a process and context switches are costly
- Threads
  - Non-isolated: avoiding incorrect interferences makes programming harder
  - Efficient: a thread is a lightweight version of a process
  - Multiple threads of control per process

Client multi-threading

Example: The Google Chrome web browser

- Multithreading: each tab runs its own thread
- A JavaScript error does not crash the main Chrome process: Only one tab freezes, the user can close it independently of others
- Takes benefit of multicore architectures, each thread runs on a separate core



Conclusion

- Concurrency has been used for decades since hardware was powerful enough to address multiple human needs

- Processes are heavy but protected whereas threads are lightweight but non-protected

- Threads require synchronization to be protected from each other

What's Next ?

- **Tutorial on Wednesday.**
  - **Multithreading**
- Read Chapter 1, 2 and 3 of the textbook
- Next week: How do we communicate in distributed systems ?
- See you all next week !