

DISTRIBUTED SYSTEMS CS6421 **ADVANCED RESOURCE MANAGEMENT**

Prof. Tim Wood and Prof. Roozbeh Haghazari

FINAL PROJECT

- Groups of 3-4 students
 - **Research-focused:** Reimplement or extend a research paper
 - **Implementation-focused:** Implement a simplified version of a real distributed system
-
- Course website has sample ideas
 - But don't feel limited by them!
 - You don't have to use go!
-
- Timeline
 - Milestone 0: Form a Team - 10/12
 - Milestone 1: Select a Topic - 10/19
 - Milestone 2: Literature Survey - 10/29
 - Milestone 3: Design Document - 11/5
 - Milestone 4: Final Presentation - 12/14

<https://gwdistsys20.github.io/project/>

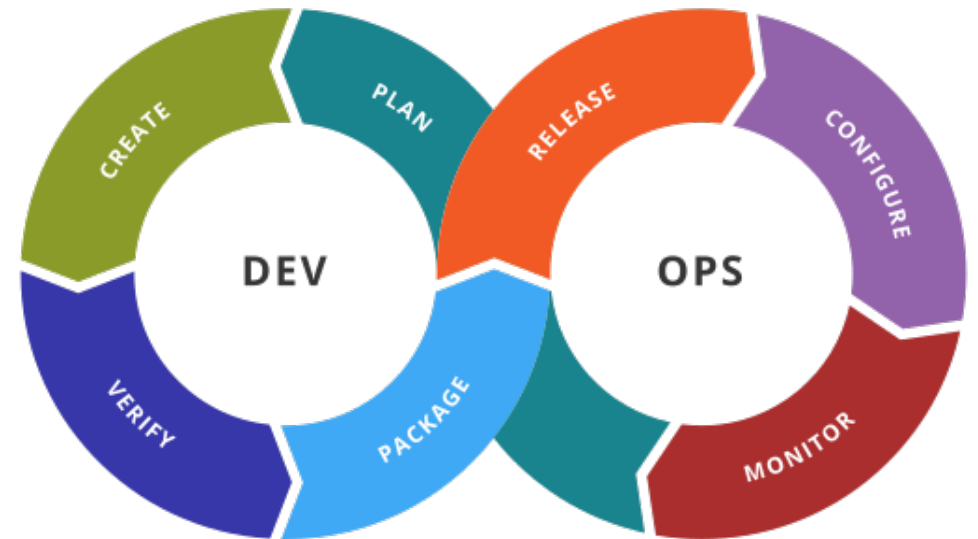
THIS WEEK...

- Case studies
 - Map reduce
 - DevOps
- Resource Optimization
 - Np-Hard problems
 - Many-Objective Optimization Problems
- Migration
 - Code
 - Processes
 - VMs
- Final Project

The future of
distributed
systems...

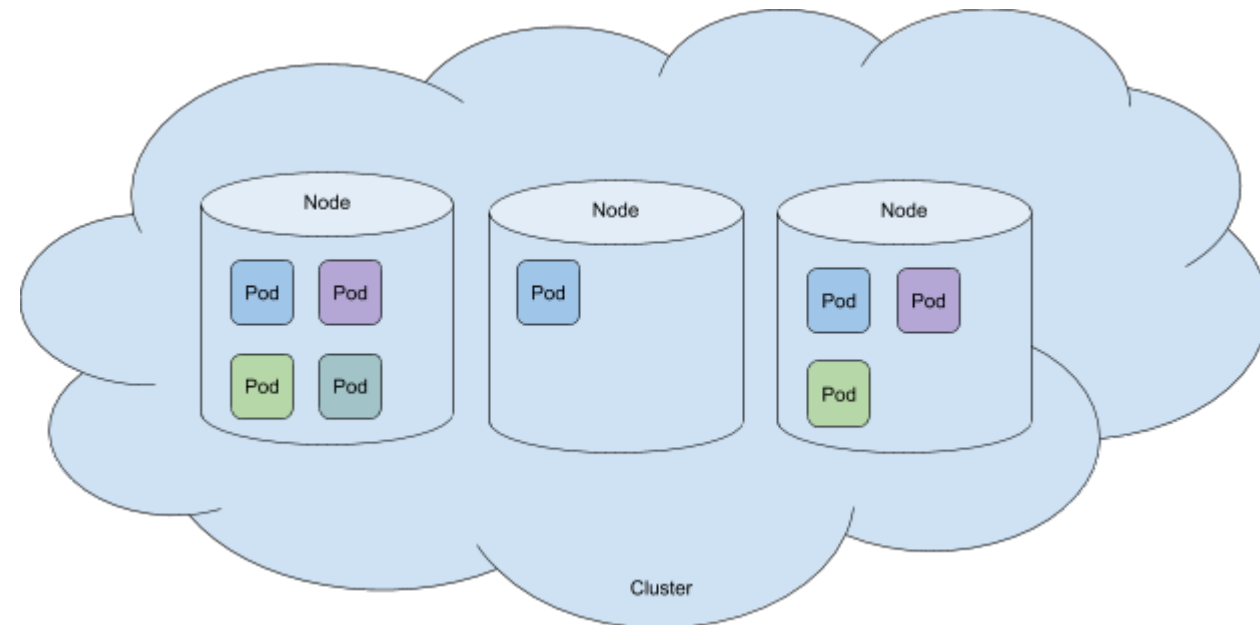
CASE STUDY: DEV OPS

- Dev Ops combines **application development** and **deployment and operations** into a single management process
- Allows companies to more quickly update and deploy applications
 - Integrates the roles of dev and ops
 - Potentially could just break things faster...
- Load Balancers have become a tool for Dev Ops to handle:
 - Service discovery
 - Health checking
 - Load balancing
 - Release management
 - ...



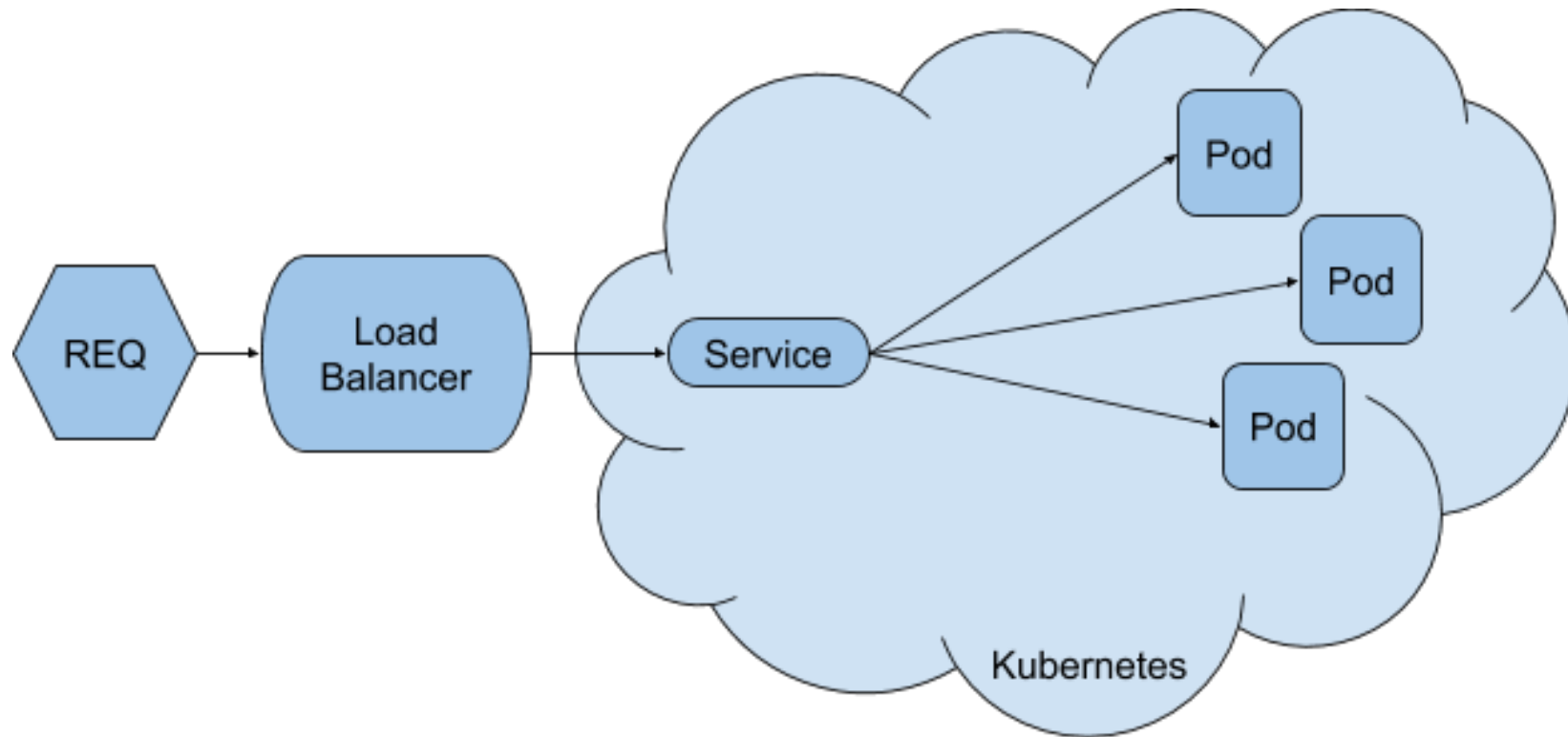
DEV OPS LB

- Kubernetes consists of physical or virtual machines—called nodes—that together form a cluster.
- Within the cluster, Kubernetes deploys pods.
- Each pod wraps a container (or more than one container) and represents a service that runs in Kubernetes. Pods can be created and destroyed as needed.
- A service is an abstraction that allows you to connect to pods in a container network without needing to know a pod's location (i.e. which node is it running on?) or to be concerned about a pod's lifecycle.

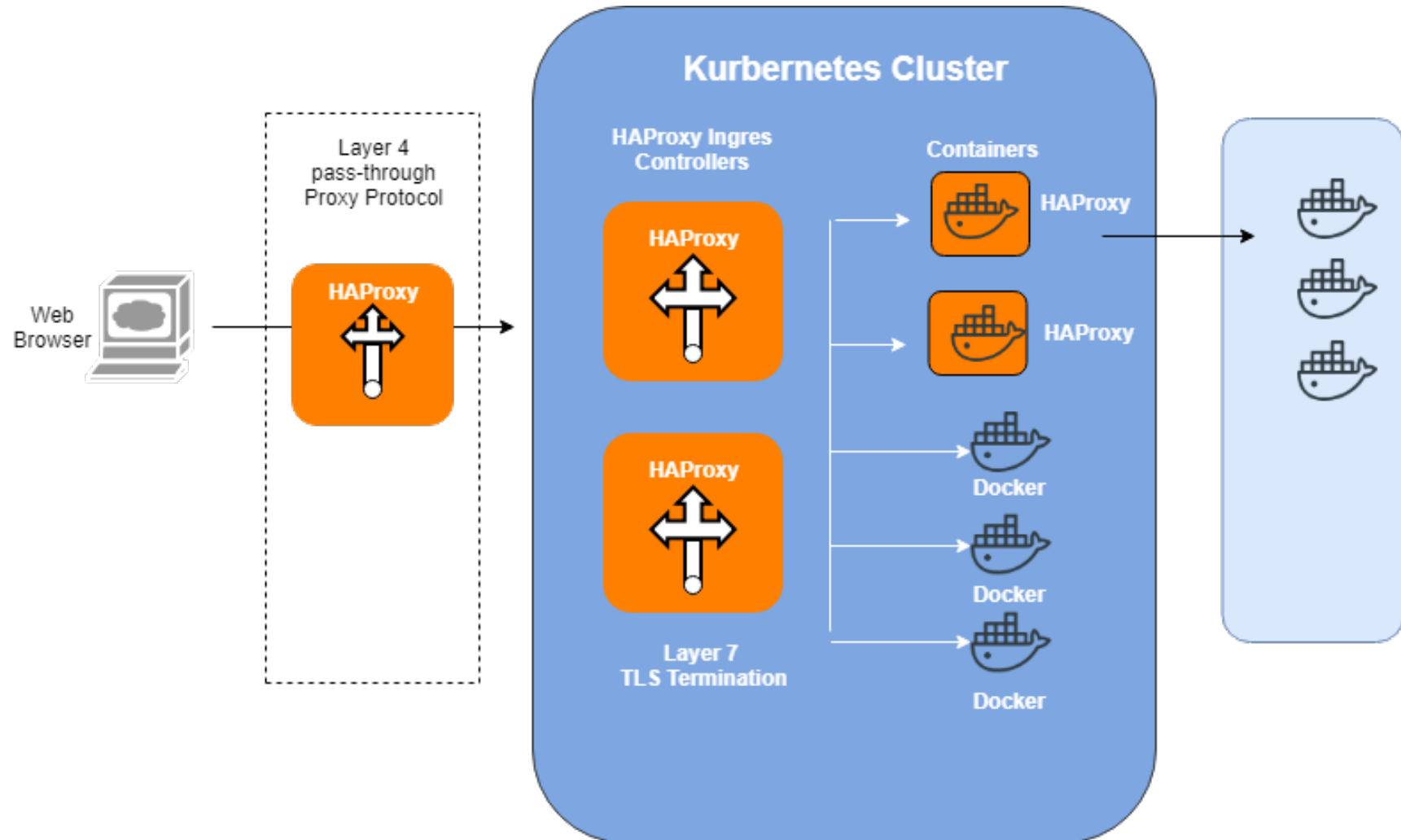


A Kubernetes cluster

DEV OPS LB



DEV OPS LB



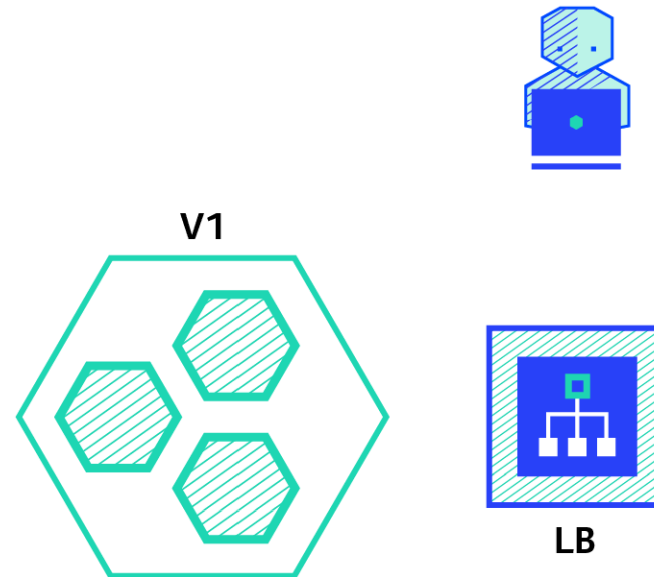
DEV OPS LB FOR DEPLOYMENT STRATEGY

- Load Balancer is just a flexible way to distribute requests
- Distribution policy doesn't need to be based on resources!
 - **Recreate:** Version A is terminated then version B is rolled out.
 - **Ramped** (also known as rolling-update or incremental): Version B is slowly rolled out and replacing version A.
 - **Blue/Green:** Version B is released alongside version A, then the traffic is switched to version B.
 - **Canary:** Version B is released to a subset of users, then proceed to a full rollout.
 - **A/B testing:** Version B is released to a subset of users under specific condition.
 - **Shadow:** Version B receives real-world traffic alongside version A and doesn't impact the response.



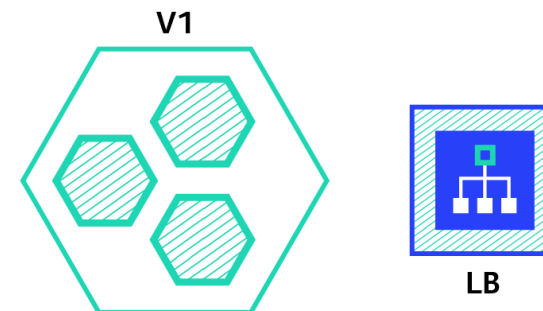
RECREATE DEPLOYMENT

- Pros:
 - Easy to setup.
 - Application state entirely renewed.
- Cons:
 - High impact on the user, expect downtime that depends on both shutdown and boot duration of the application.



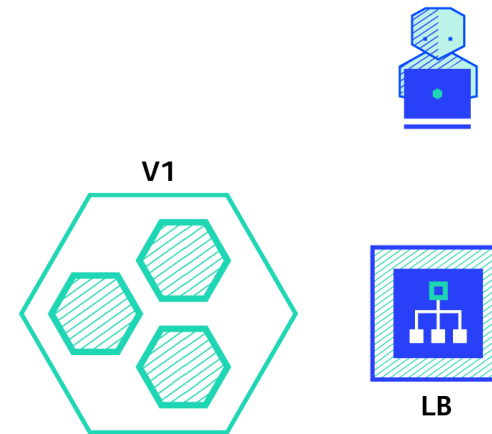
RAMPED

- When an instance of pool B is deployed and its service would be ready, one instance from pool A would be shut down.
- Depending on the system taking care of the ramped deployment, you can tweak the following parameters to increase the deployment time:
 - Parallelism, max batch size: Number of concurrent instances to roll out.
 - Max surge: How many instances to add in addition of the current amount.
 - Max unavailable: Number of unavailable instances during the rolling update procedure.



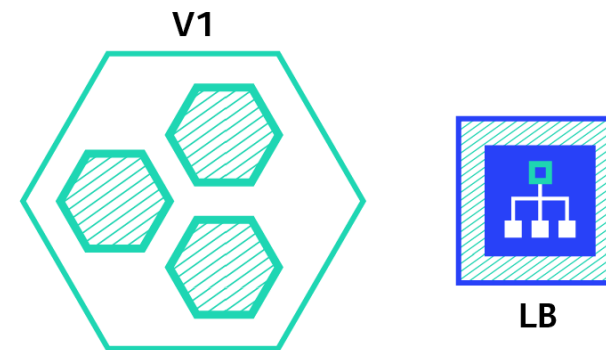
BLUE/GREEN

- The blue/green deployment strategy differs from a ramped deployment, version B (green) is deployed alongside version A (blue) with exactly the same amount of instances. After testing that the new version meets all the requirements the traffic is switched from version A to version B at the load balancer level.



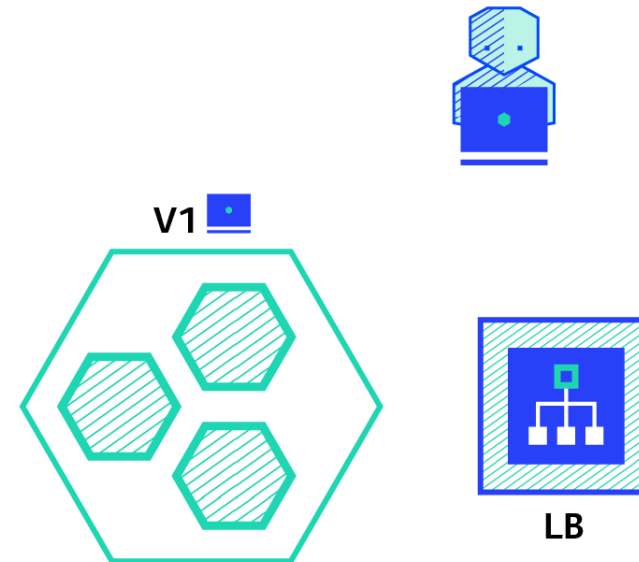
CANARY

- A canary deployment consists of gradually shifting production traffic from version A to version B. Usually the traffic is split based on weight.



A/B TESTING

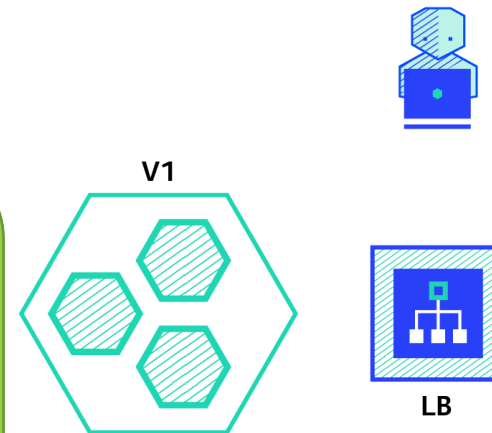
- A/B testing deployments consists of routing a subset of users to a new functionality under specific conditions. It is usually a technique for making business decisions based on statistics, rather than a deployment strategy.
- Here is a list of conditions that can be used to distribute traffic amongst the versions:
 - By browser cookie
 - Query parameters
 - Geolocation
 - Technology support: browser version, screen size, operating system, etc.
 - Language



SHADOW

- A shadow deployment consists of releasing version B alongside version A, fork version A's incoming requests and send them to version B as well without impacting production traffic.
- This is particularly useful to test production load on a new feature. A rollout of the application is triggered when stability and performance meet the requirements.

For example, given a shopping cart platform, if you want to shadow test the payment service you can end-up having customers paying twice for their order.



SCHEDULING IN MAP REDUCE

- Researchers have considered many factors when designing big data scheduling algorithms:
 - What types of factors might we care about for MR scheduling?

SCHEDULING IN MAP REDUCE

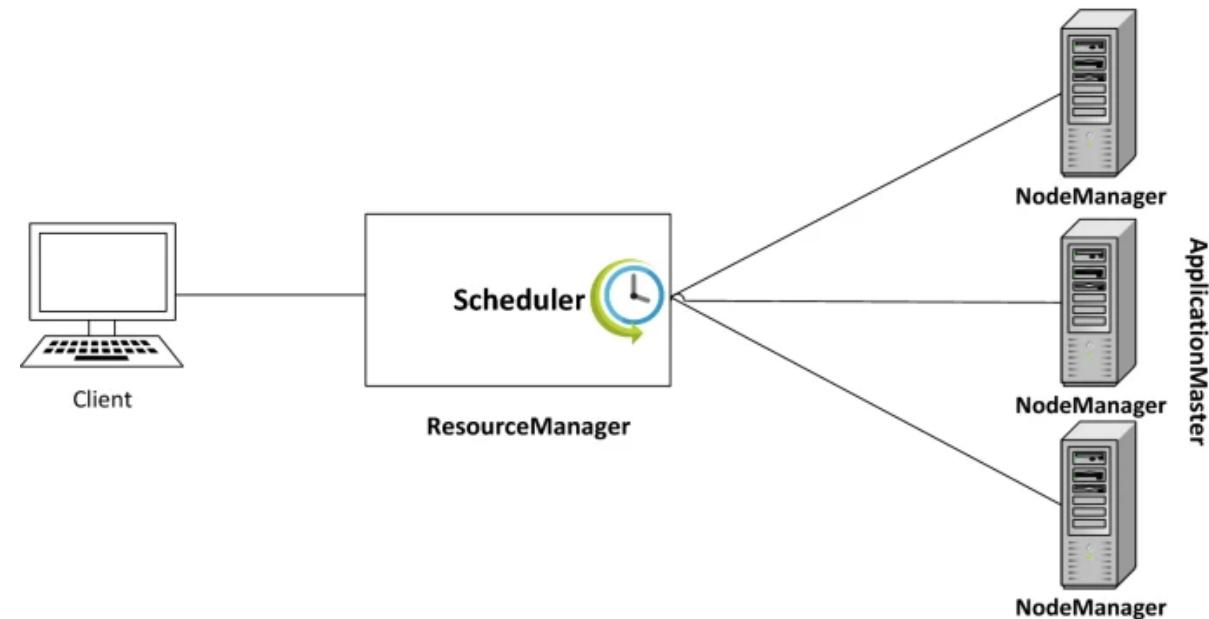
- Researchers have considered many factors when designing big data scheduling algorithms:
 - Resource Efficiency
 - Data Locality
 - Deadlines
 - Hardware and Task Heterogeneity
 - Nature of jobs (dependencies, discrete or continuous problem space)
 - Energy consumption
 - Latency of short tasks vs throughput of big tasks

BASIC MAP REDUCE TASK SCHEDULING

- **FIFO** - Assigns resources to jobs based on arrival time.
 - Fully complete one job before starting the next
- **Fair** - Assigns resources to jobs so that all jobs get an equal share of resources over time
 - Splits up cluster to run multiple jobs simultaneously
 - Jobs are grouped into pools (e.g., all jobs from one user are in the same pool)
 - Fairness is provided across pools; jobs within a pool can be FIFO or Fair
- **Capacity** - Assigns resources to jobs based on its organization's capacity
 - Each organization contributes resources to the cluster, guaranteeing its minimum share
 - If an organization is not using all resources, others can use them in a fair manner
 - Supports priorities, security ACLs, and resource requirements (only RAM)

YARN MAP REDUCE TASK SCHEDULING

- Hadoop Yarn is a framework, which provides a management solution for big data in distributed environments.
- Provides support for:
 - multi-tenant environment
 - cluster utilization
 - high scalability
 - implementation of security controls
- Yarn consists of two main components which are:
 - Resource Manager
 - Application master

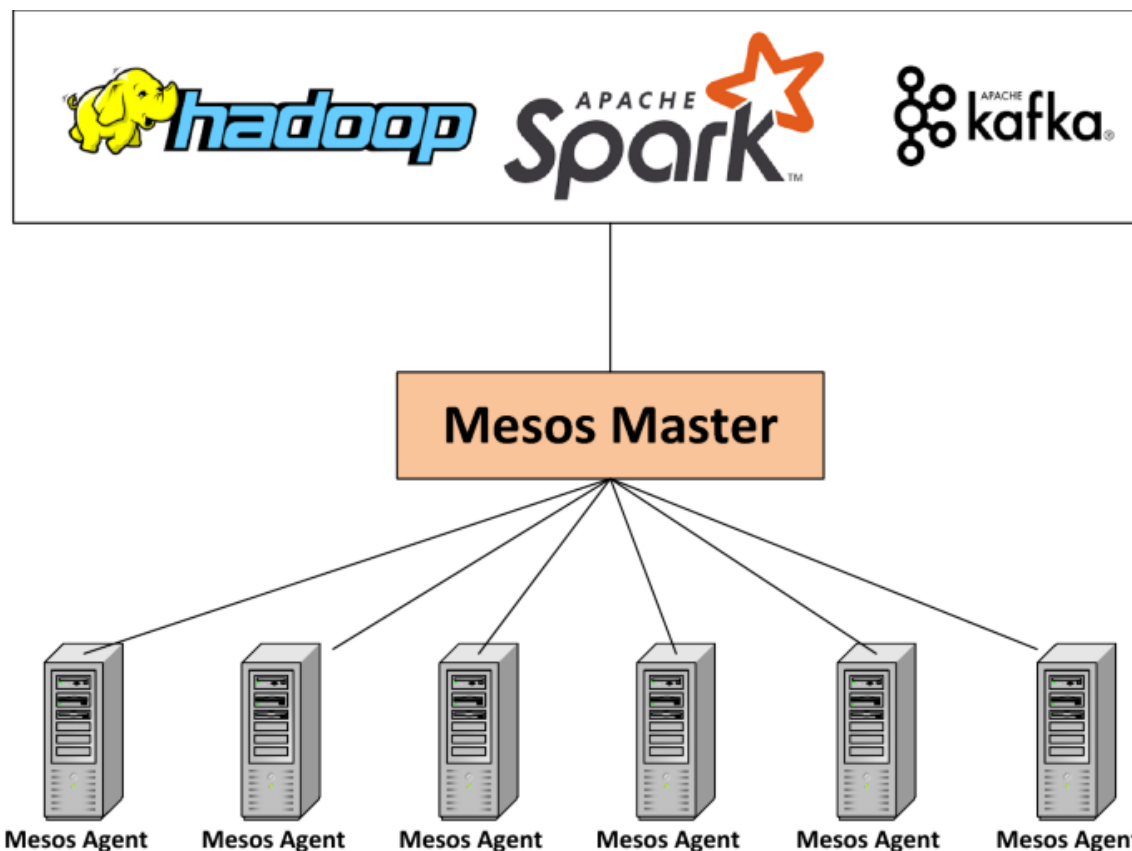


CORONA

- Corona is an extension of the MapReduce framework from Facebook
- It provides high scalability and cluster utilization for small tasks.
- This extension was designed to overcome some of the important Facebook challenges, such as:
 - Scalability
 - Low latency for small jobs (pull-model)
 - Resource requirements
 - Dynamic software updates
- Introduces more scalable job tracking and scheduling components

More info: <https://www.facebook.com/notes/facebook-engineering/under-the-hood-scheduling-mapreduce-jobs-more-efficiently-with-corona/10151142560538920/>

APACHE MESOS



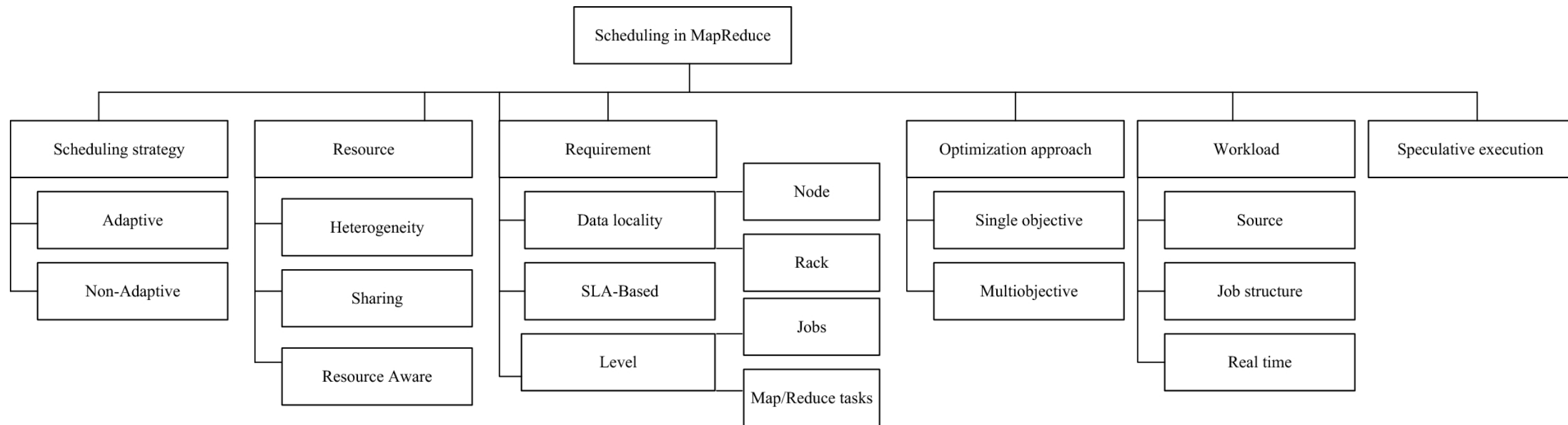
- Cluster manager to offer effective heterogeneous resources isolation and allocation for distributed applications
 - Originally developed at UC Berkeley, extended at Twitter/AirBnB/others
- Defines an abstraction of computing resources (CPU, storage, network, memory, and file system)
- Supports customizable schedulers that match requests from applications to cluster resources
 - Not MapReduce/Hadoop specific

RESOURCE SCHEDULING FRAMEWORKS

| Features | MapReduce default [21] | Yarn [22] | Mesos [23] | Corona [24] |
|---------------------|------------------------|---------------|----------------|-----------------|
| Resources | Request based | Request based | Offer based | Push based |
| Scheduling | Memory | Memory | Memory/CPU | Memory/CPU/Disk |
| Cluster utilization | Low | High | High | High |
| Fairness | No | Yes | Yes | Yes |
| Job latency | High | Low | Low | Low |
| Scalability | Medium | High | High | High |
| Computation model | Job/task based | Cluster based | Cluster based | Slot based |
| Language | Java | Java | C++ | – |
| Platform | Apache Hadoop | Apache Hadoop | Cross-platform | Cross-platform |
| Open source | Yes | Yes | Yes | Yes |
| Developer | ASF | ASF | ASF | Facebook |

TAXONOMY OF MAPREDUCE SCHEDULING

A **taxonomy** helps us structure our comparisons of different categories of MapReduce Schedulers



MAP REDUCE SCHEDULING

- Continues to evolve over time as needs change
 - Scale of MR clusters
 - Diversity of users and workloads
 - Size of data to process
 - Hardware accelerators
- Might be a good area for a Final Project!

ADVANCED RESOURCE MANAGEMENT ALGORITHMS



PROBLEM DEFINITIONS

- $O(1)$ – constant-time
- $O(\log_2(n))$ – logarithmic-time
- $O(n)$ – linear-time
- $O(n^2)$ – quadratic-time
- $O(n^k)$ – polynomial-time
- $O(k^n)$ – exponential-time
- $O(n!)$ – factorial-time

POLYNOMIAL ALGORITHMS

- $T(n) = (C * n^k)$ where $C > 0$ and $k > 0$ where C, k are constant and n is input size
- In general, for polynomial-time algorithms k is expected to be less than n .
- Many algorithms complete in polynomial time:
 - All basic mathematical operations; addition, subtraction, division, multiplication
 - Testing for primacy
 - Hash-table lookup, string operations, sorting problems
 - Shortest Path Algorithms; Dijkstra, Bellman-Ford, Floyd-Warshall
 - Linear and Binary Search Algorithms for a given set of numbers

NP ALGORITHMS

- Cannot be solved in polynomial time. However, they can be verified (or certified) in polynomial time. (verification can be done by Turing machine)
- We expect these algorithms to have an exponential complexity, which we'll define as:
- $T(n) = (C_1 * k^{C_2 * n})$ where $C_1 > 0$, $C_2 > 0$ and $k > 0$ where C_1, C_2, k are constant and n is input size
- complexity is $O(k^n)$ for some k and their results can be verified in polynomial time.

- Salesman Problem
- Integer Factorization
- Graph Isomorphism

NP-COMPLETE ALGORITHMS

- What makes them different from other NP problems is a useful distinction called completeness.
 - Traveling Salesman
 - Knapsack
 - Graph Coloring

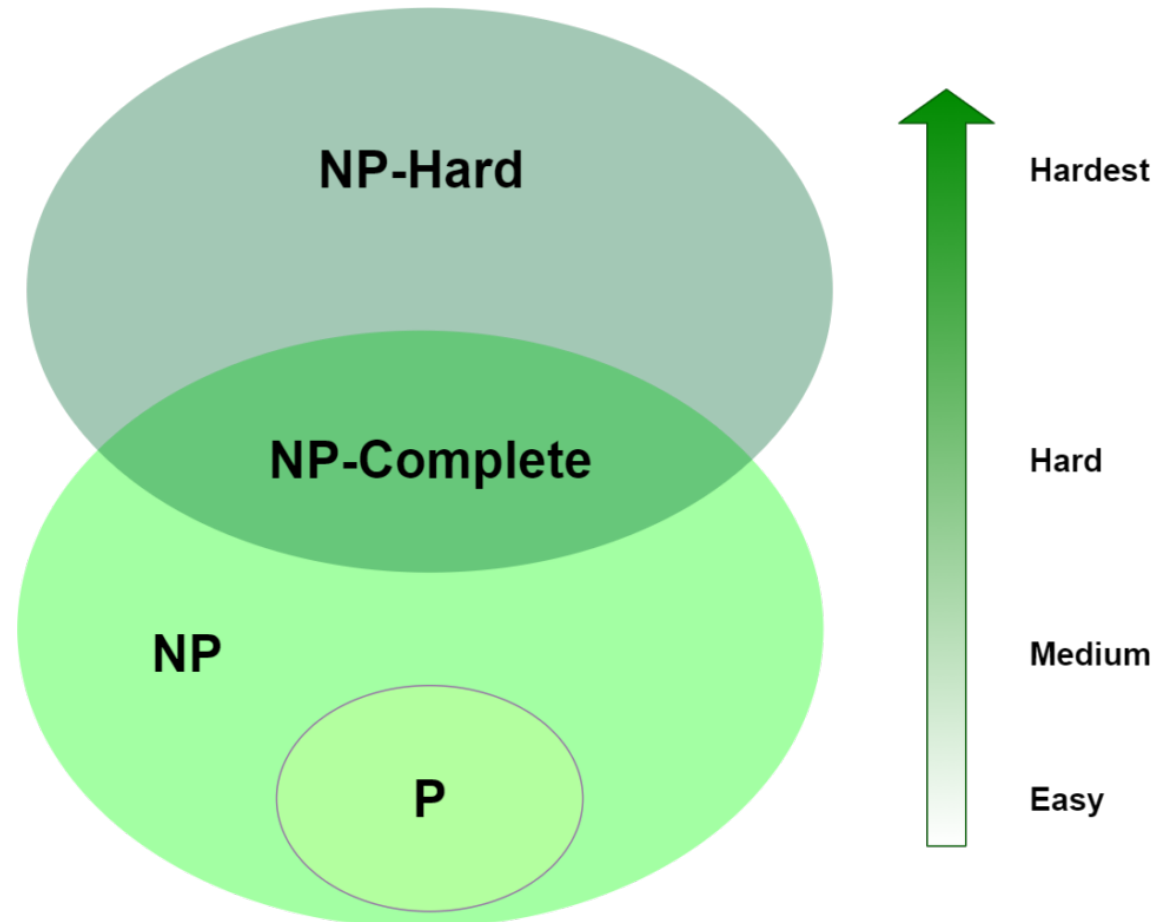
- Prove $P=NP$ or $P \neq NP$

<https://www.claymath.org/millennium-problems/p-vs-np-problem>

https://en.wikipedia.org/wiki/Millennium_Prize_Problems

NP-HARD ALGORITHMS

- Non-deterministic Polynomial-time hard
- Most complex problems in computer science.
- They are not only hard to solve but are hard to verify as well.
 - K-means Clustering
 - Traveling Salesman Problem,
 - Graph Coloring
 - maximum clique problem
- These algorithms have a property similar to ones in NP-Complete – they can all be reduced to any problem in NP



MANY OBJECTIVE OPTIMIZATION PROBLEM

- Many objective problems are the problems that have some objectives which should be satisfied by the solutions.

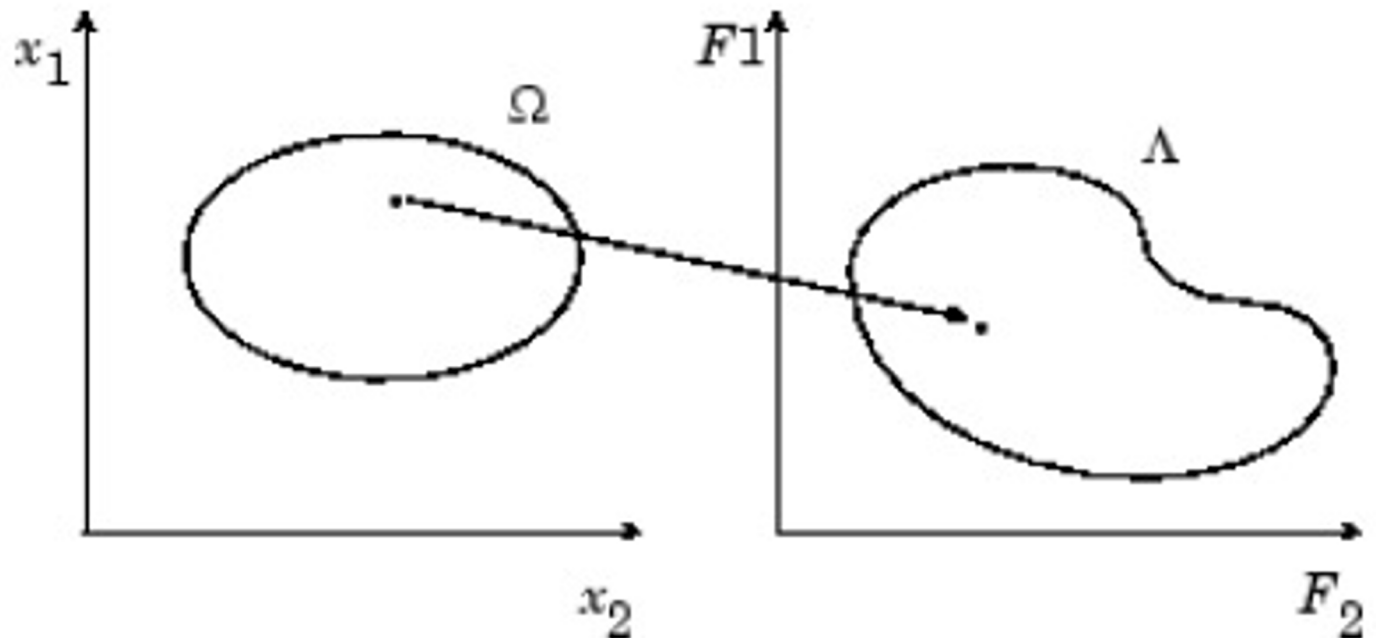
find $x \in \phi: f(x) \leq f(y), \forall y \in \phi$

$$\min_x f(x) \in R, x \in \phi$$

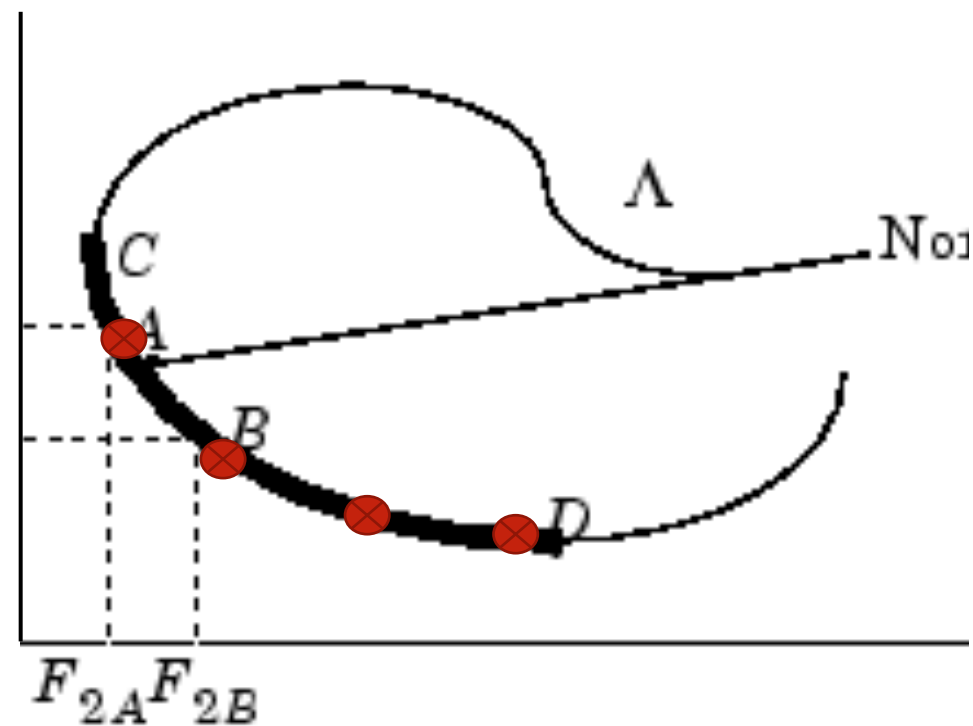
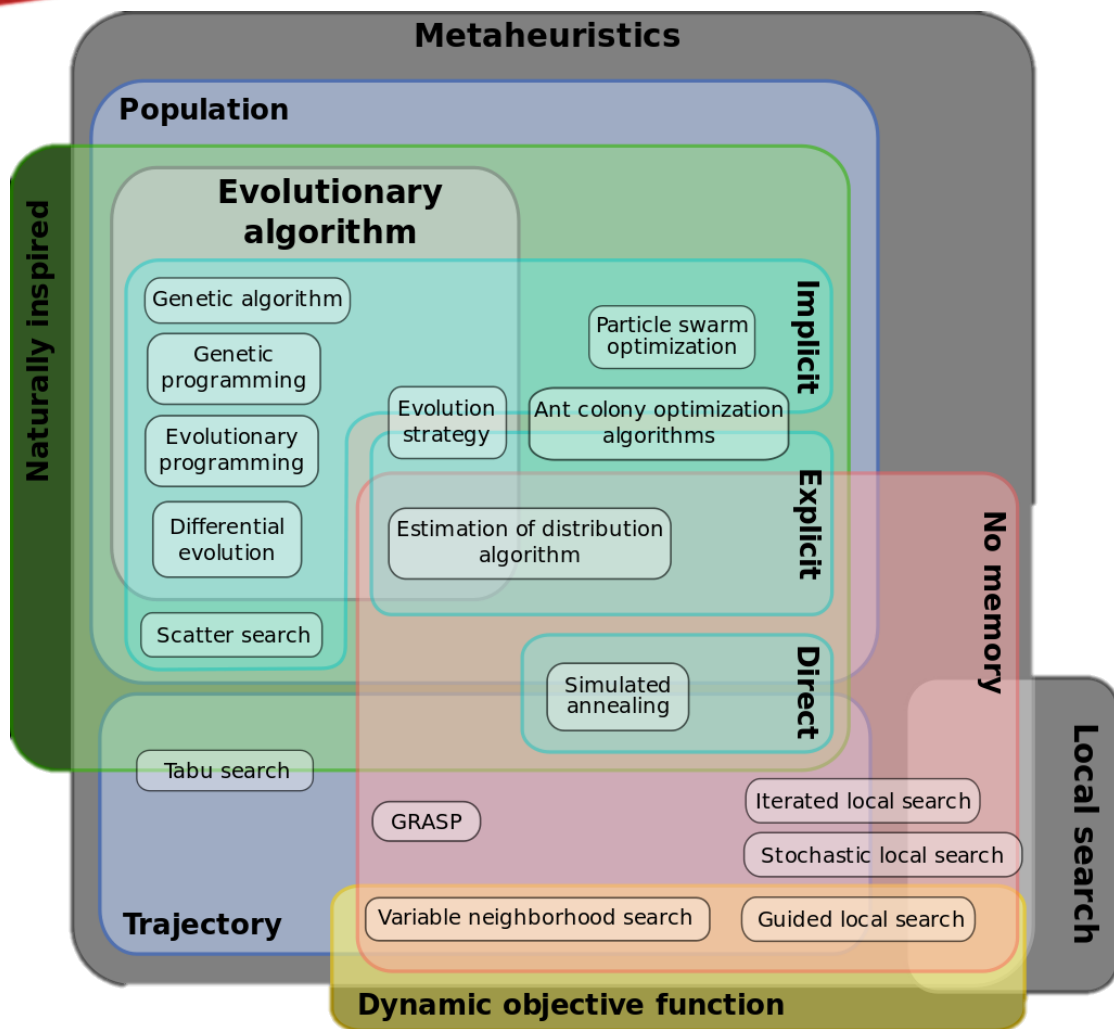
$$f: X \subset R^n \rightarrow Y \subset R$$

$$X = \{x = (x_1 \dots x_n), x_i \in D_i\}$$

$$\phi = \begin{cases} g_i(x) \leq 0 \\ h_j(x) = 0 \\ x \in X \end{cases}$$

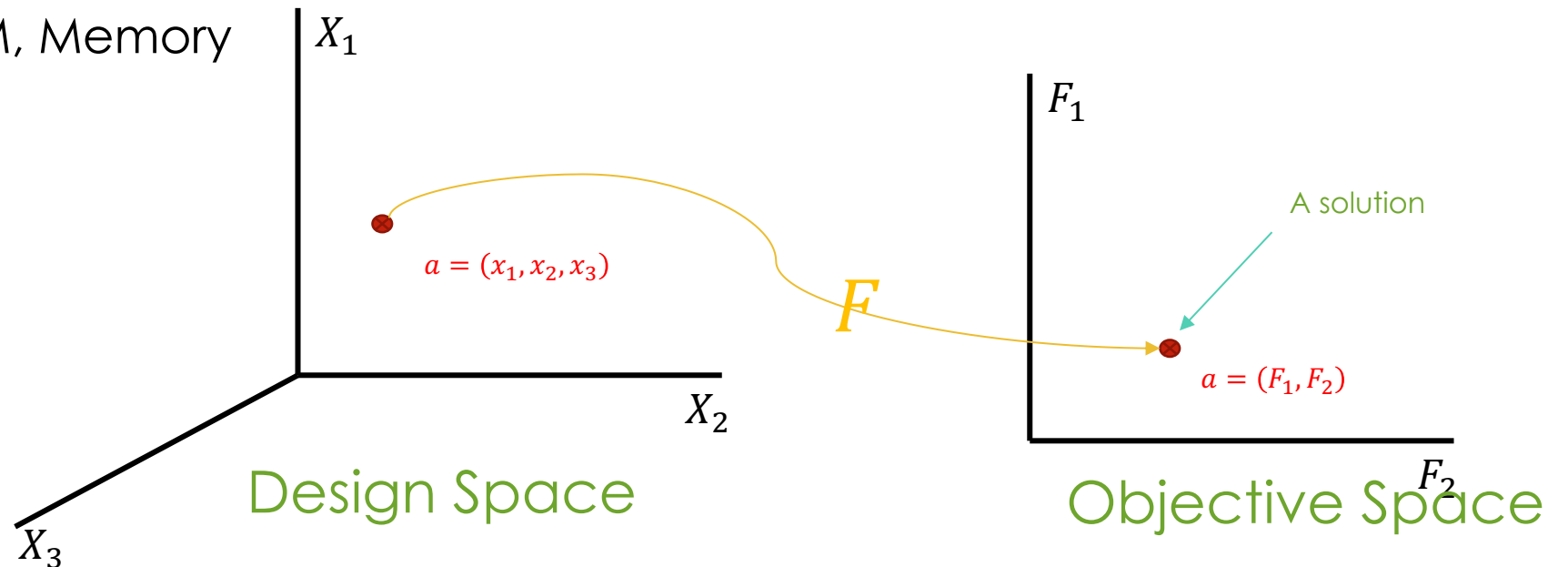


FINDING THE PARETO FRONT



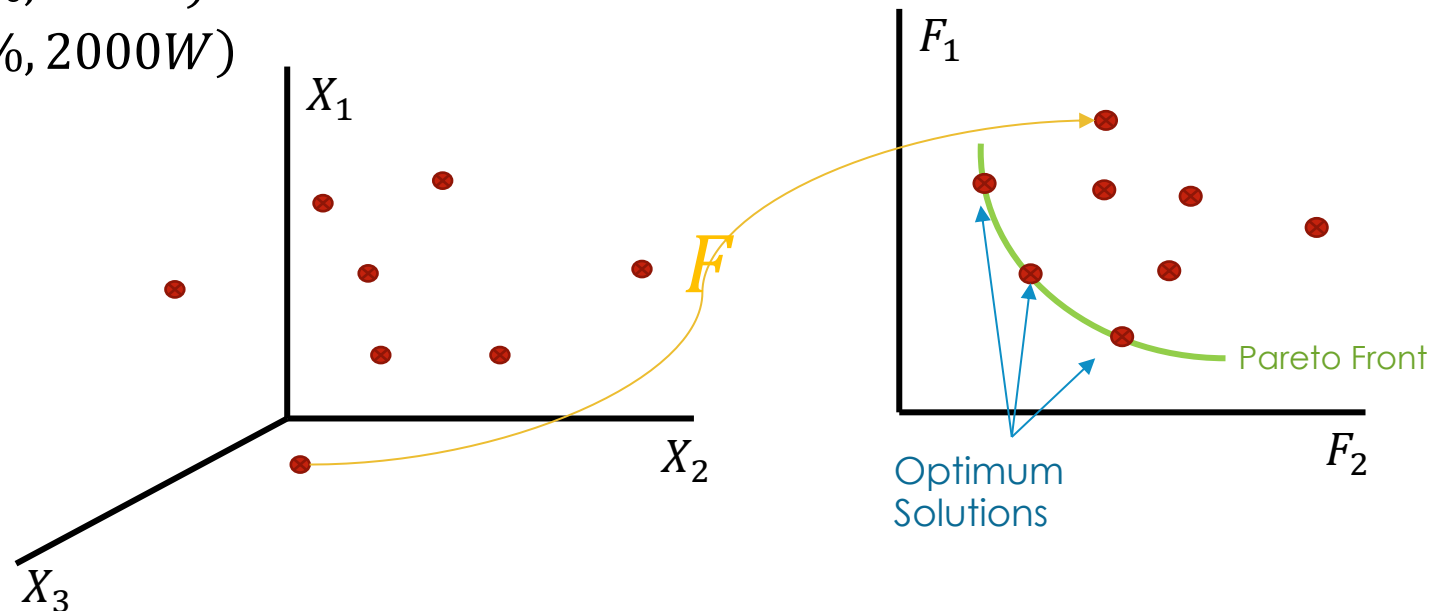
PLACEMENT AS AN OPTIMIZATION PROBLEM

- $f: X \subset R^n \rightarrow Y \subset R$
- $X = \{X_1: VM1, X_2: VM2, X_3: VM3\}$
- $Y = \{F_1: Utilization, F_2: Power Consumption\}$
- Constrains: CPU, RAM, Memory



PLACEMENT AS AN OPTIMIZATION PROBLEM

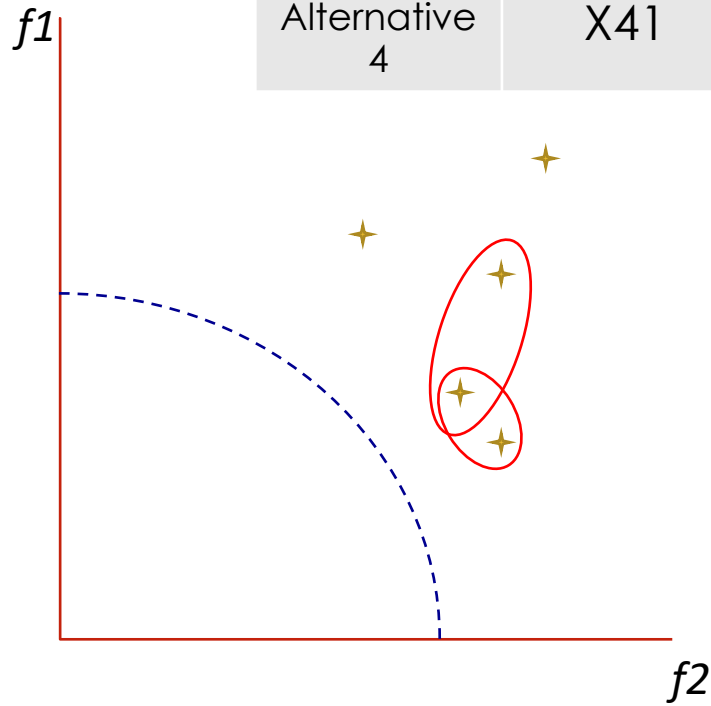
- For example we can see solutions like this when we aim to find the best placement of our VMs among $VM_1 \dots VM_{10}$:
 - $a_1 = (VM_1, VM_5, VM_6) \rightarrow (30\%, 900W)$
 - $a_2 = (VM_1, VM_3, VM_7) \rightarrow (70\%, 1100W)$
 - $a_3 = (VM_2, VM_8, VM_1) \rightarrow (50\%, 950W)$
 - $a_4 = (VM_2, VM_8, VM_1) \rightarrow (85\%, 2000W)$
 - ...



MULTI CRITERIA DECISION MAKING

- Choose the best option among the alternatives
- Strategies
 - Pair-wise Comparison
 - AHP, ANP, ELECTRE...
 - Reference Distance
 - VIKOR, TOPSIS,

| | Criterion 1 | Criterion 2 | Criterion 3 | Criterion 4 |
|---------------|-------------|-------------|-------------|-------------|
| Alternative 1 | X11 | X12 | X13 | X14 |
| Alternative 2 | X21 | X22 | X23 | X24 |
| Alternative 3 | X31 | X32 | X33 | X34 |
| Alternative 4 | X41 | X42 | X43 | X44 |



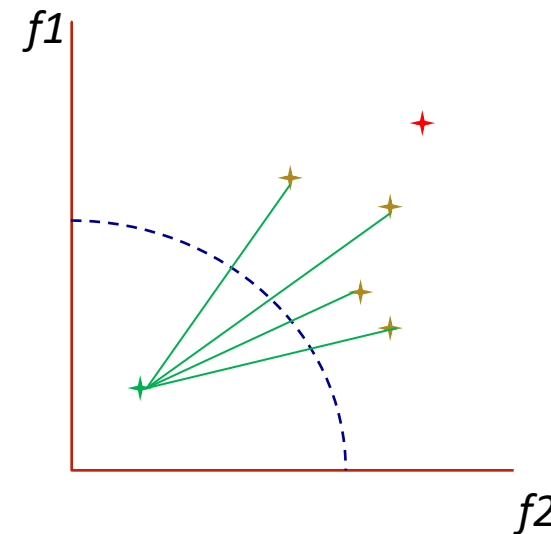
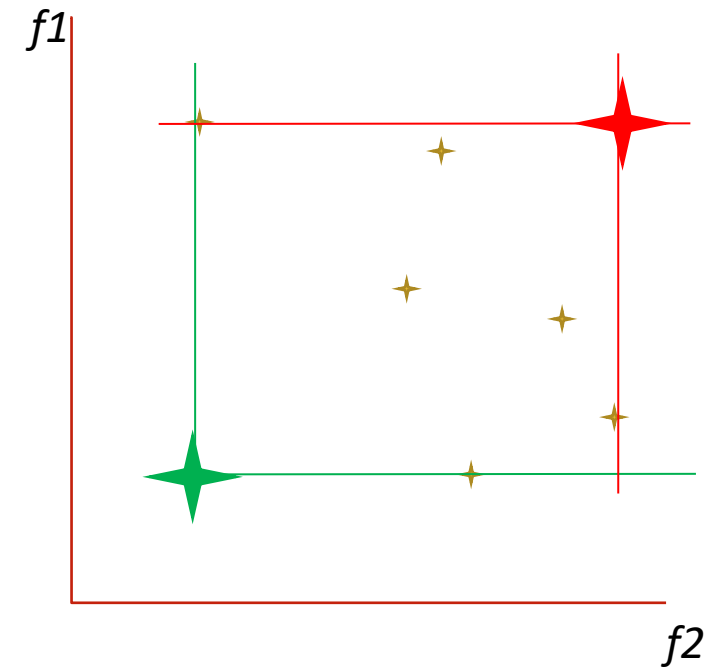
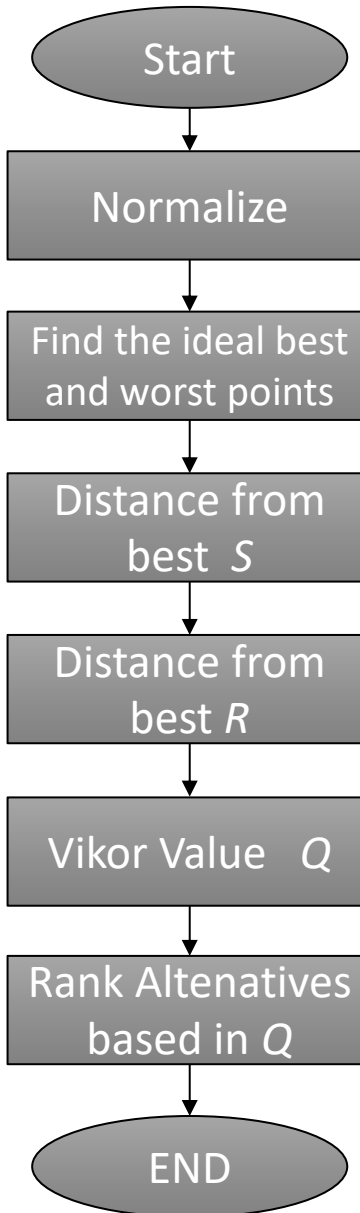
MCDM

- For example: VIKOR is one of the MCDM methods which:
 - The complexity of the algorithm is low
 - Selection is based on the distance to the best point

$$S_i = \sum_{j=1}^n w_j \frac{(f_j^* - f_{ij})}{(f_j^* - f_j^-)}$$

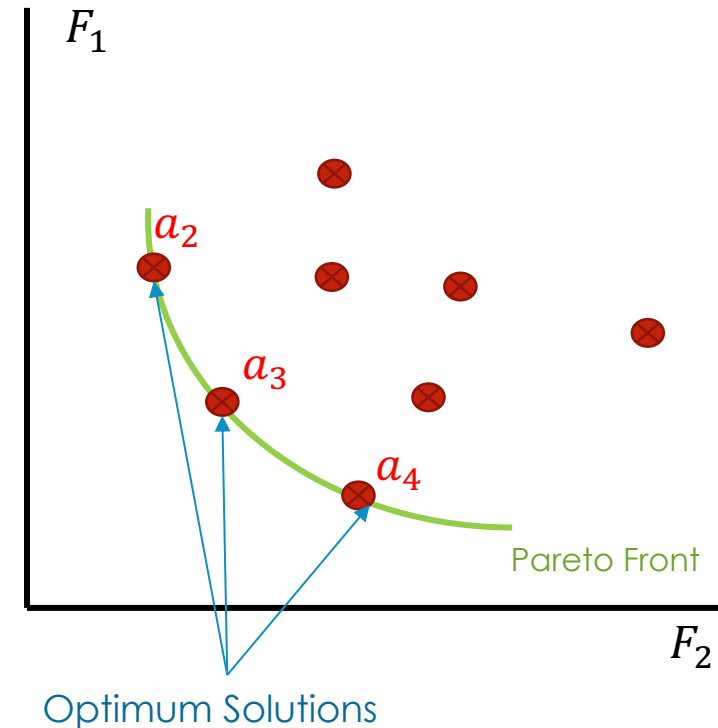
$$R_i = \text{MAX}[w_j \frac{(f_j^* - f_{ij})}{(f_j^* - f_j^-)}]$$

$$Q_i = v \left[\frac{S_i - S^*}{S^- - S^*} \right] + (1 - v) \left[\frac{R_i - R^*}{R^- - R^*} \right]$$



PLACEMENT AS A DECISION MAKING PROBLEM

| | Criterion1 | Criterion2 | Criterion3 | Criterion4 |
|-------|------------|------------|------------|------------|
| a_4 | X11 | X12 | X13 | X14 |
| a_3 | X21 | X22 | X23 | X24 |
| a_2 | X31 | X32 | X33 | X34 |



DYNAMIC PLACEMENT WITH CODE MIGRATION



CODE MIGRATION

Placement doesn't need to be **static**

Migration allows us to move code between nodes on demand

Why? What? How?

YOU USE MIGRATION EVERY DAY!

- _____ is the simplest and most common form of code migration
 - You probably use this hundreds of times every day...
- **What is it?**
- **Why is it used?**
- **What problems does it cause?**

YOU USE MIGRATION EVERY DAY!

- JavaScript is the simplest and most common form of code migration
 - You probably use this hundreds of times every day...
- JavaScript files (source code) are migrated from the server to your web browser for execution
 - More responsive UI, local error checking
 - Takes advantage of client processing power
 - Can open security vulnerabilities
 - Adds complexity to software

WHY MIGRATE?

- The ability to move where code is executed provides flexibility
- Performance
 - Reduced perceived latency by having JS update a GUI locally
- Resource efficiency
 - Move VMs to a different server for consolidation
- Security
 - Move processing to data for compliance reasons

MIGRATION CHARACTERISTICS

- **Weak Mobility:** just the code is moved and restarted
 - e.g., JavaScript.
 - Simple implementation, but limited flexibility
- **Strong Mobility:** code & state is moved and execution continues seamlessly
 - e.g., VM migration
 - Very powerful, but hard to implement
- Which side of the communication starts the migration?
 - The machine currently executing the code (**sender-initiated**)
 - The machine that will ultimately execute the code (**receiver-initiated**).

WHAT TO MIGRATE?

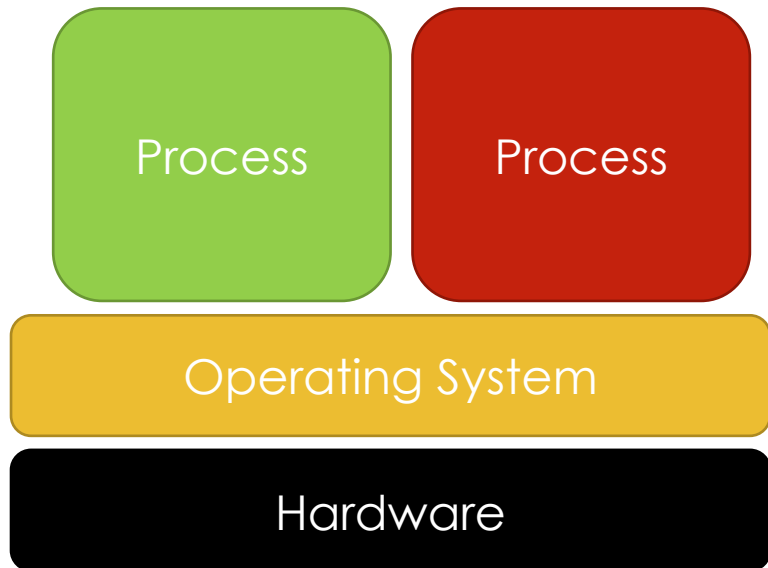
- A running component consists of three “segments”:
 1. Code – instructions
 2. Resources – external references
 3. Execution – current state

JavaScript code migration?

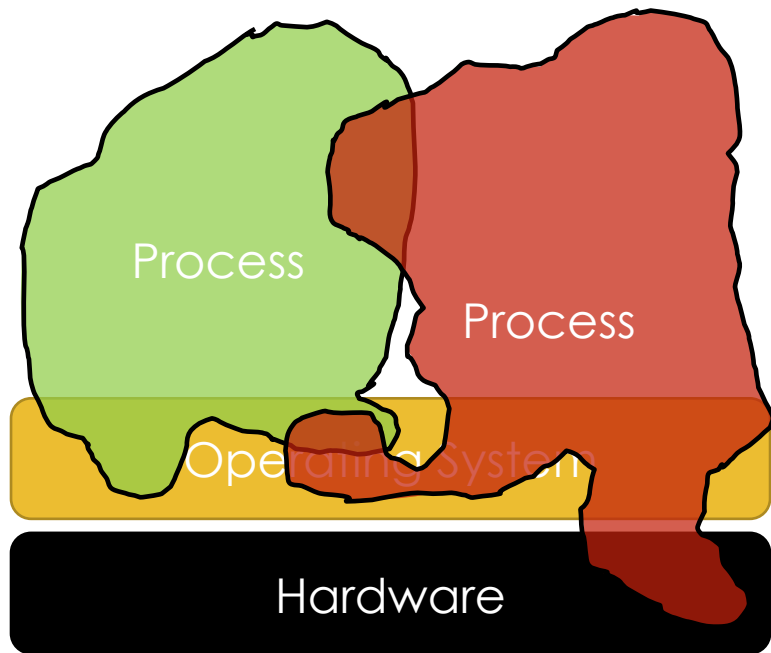
Process migration?

HOW TO MIGRATE A PROCESS?

- How cleanly isolated is a process?



HOW TO MIGRATE A PROCESS?



- Code, libraries, data/image files
- Stack/Heap/registers
- OS state: file descriptors, sockets, scheduler information, permissions

- IP address?
- HW devices like GPUs, printers?
- Inter-process communication?
- Requirements for destination?

WHAT ABOUT RESOURCES?

- What makes code migration difficult is the requirement to migrate resources.
- Resources are the external references that a process is currently using, and includes (but is not limited to):
 - Variables, open files, network connections, printers, databases, etc...
- Not all resources can be migrated
- What if source and destination hosts are heterogenous?

PROCESS MIGRATION WITH CRIU

Pronounced Kree-ew

- CRIU = Checkpoint/Restore in Userspace
 - Libraries supported by Linux to enable checkpointing and migration
 - “In userspace” meaning it requires limited kernel support
- Checkpoint + Restore = Migration!

- Saves all process and OS state to a file
 - Can operate on a tree of processes (containers!)
- Restore process(es) from a checkpoint file

- Based on old research like [Zap, OSDI 2002]



VM MIGRATION

Supported by Xen, Vmware, KVM

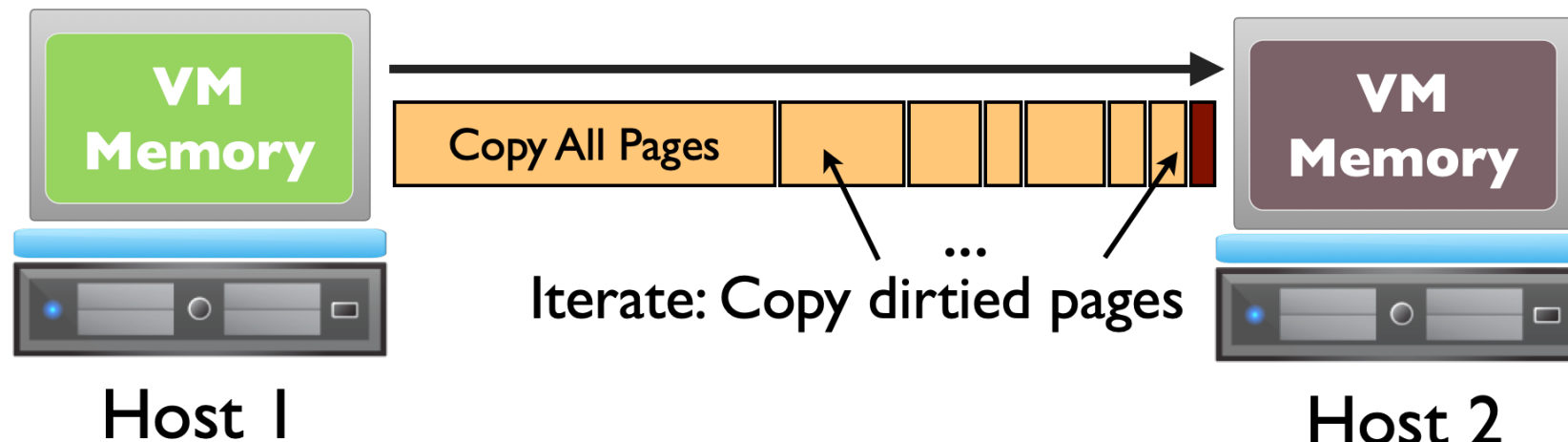
- Moving an entire VM is actually a lot simpler!
- Virtual machine cleanly encapsulates all processes and OS
 - Simple, well defined interfaces are very useful in systems!
- Migrate VM's memory and CPU state
- Update network configuration (ARP message)



Trade-offs?

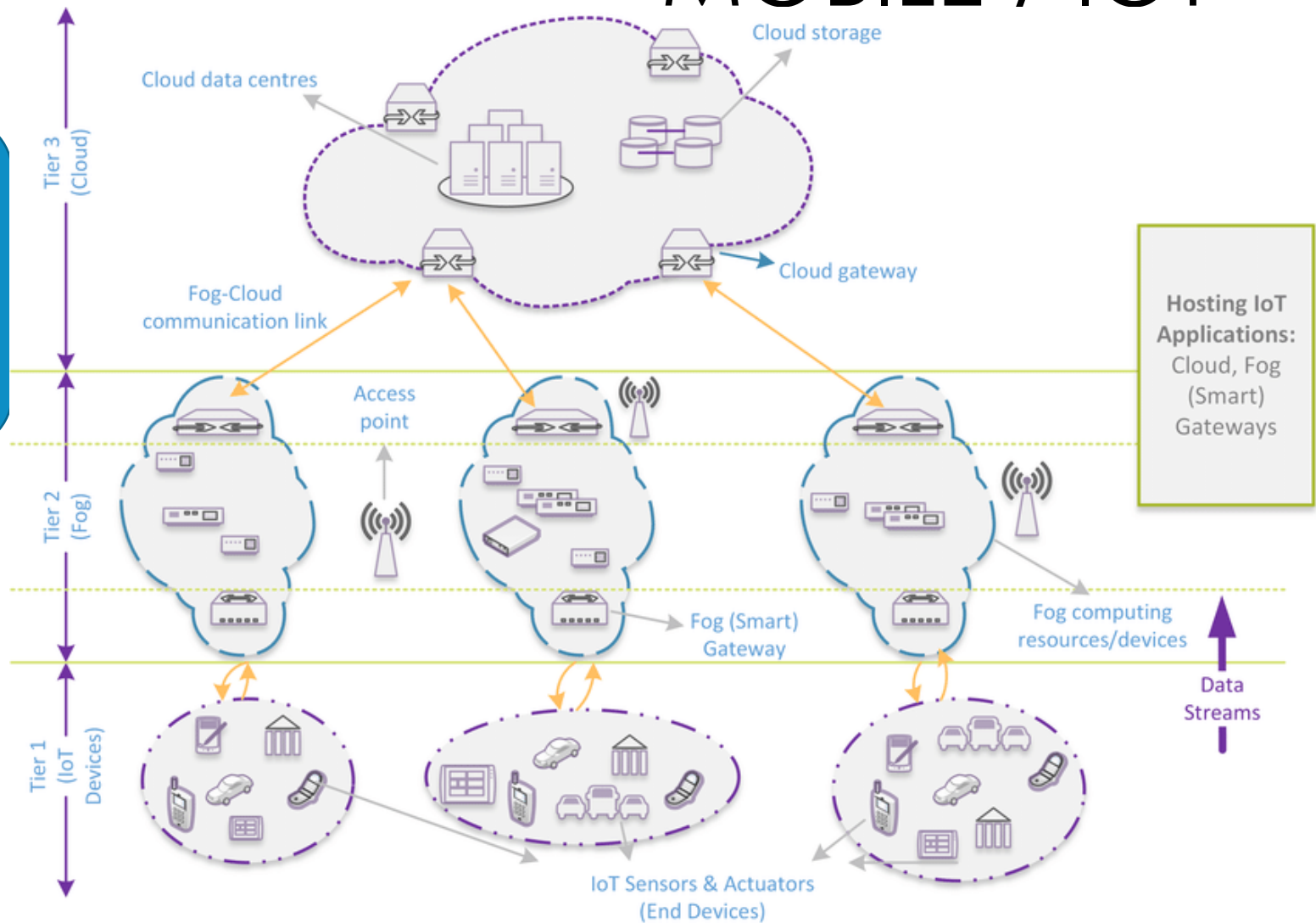
VM MIGRATION TECHNIQUES

- **Stop and Copy:** pause VM and copy all of its data, then resume on host 2
- **Iterative:** Copy pages as VM runs, track what gets “dirtyed” and resend
- **Pull:** Start running on destination immediately and pull missing pages over network on demand



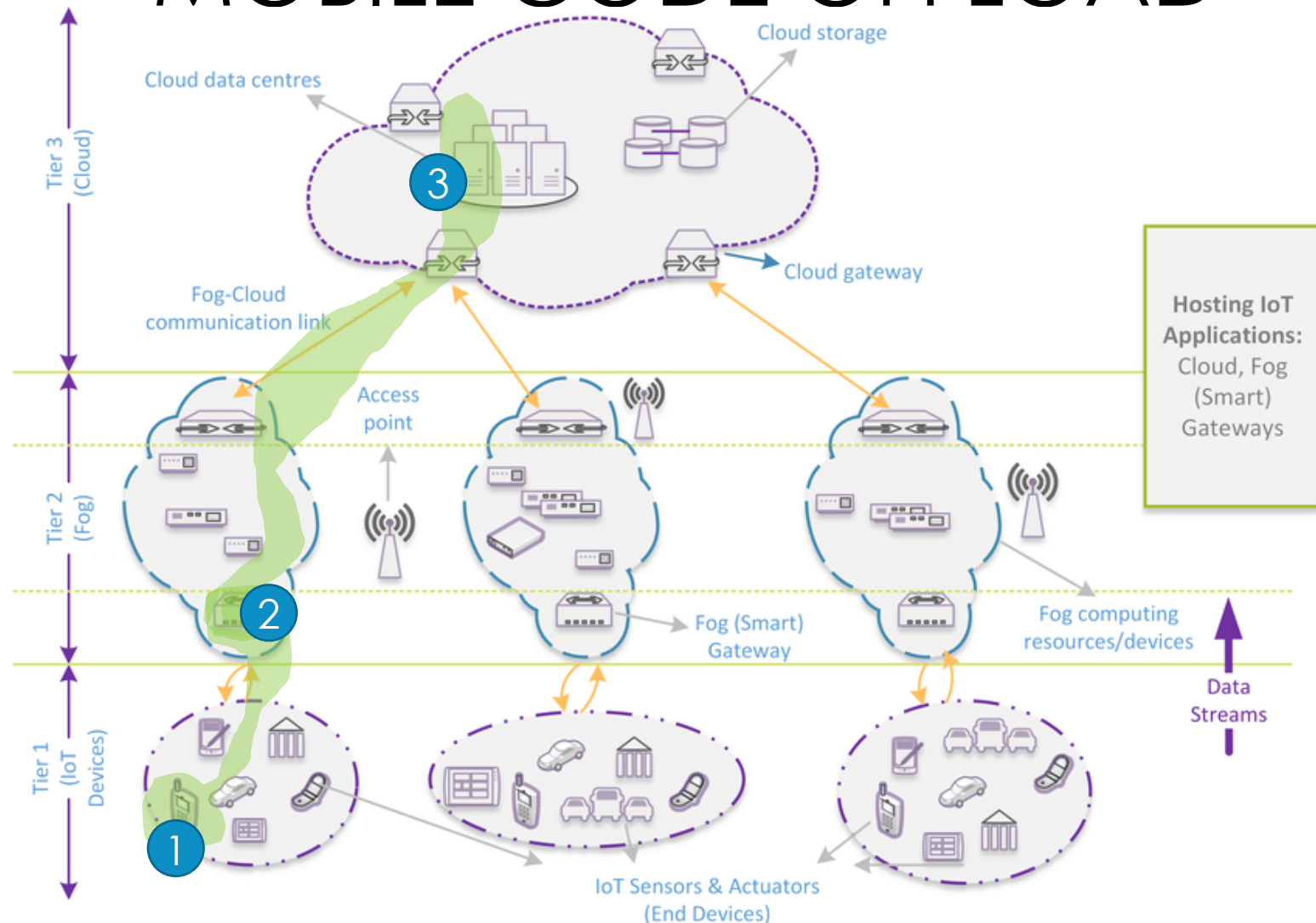
MOBILE / IOT

How can we use migration in a mobile / IoT environment?



MOBILE CODE OFFLOAD

- Mobile Code Offload
 - Migrate components of an application between phone and cloud
- Decide what to migrate based on available resources
 - Network latency to cloud?
 - Battery life on device?
- See [Maui, MobiSys 2010] and others



SOFTWARE AGENTS

- What is a software agent?
 - “An **autonomous** unit capable of performing a task in collaboration with other, possibly remote, agents”.
- Software agents are a software architecture that focuses on dynamic, flexible software components that can make their own decisions
 - Can involve dynamic migration driven by the software component itself
- Autonomic Computing
 - Self-configuring, Self-managing, Self-healing, Self-optimizing, Self-organizing...
 - Goal is to reduce the complexity of distributed systems by building intelligence and automated control into the components

FINAL PROJECT

- Groups of 3-4 students
- **Research-focused:** Reimplement or extend a research paper
- **Implementation-focused:** Implement a simplified version of a real distributed system
- Course website has sample ideas
 - But don't feel limited by them!
 - You don't have to use go!
- Timeline
 - Milestone 0: Form a Team - 10/12
 - Milestone 1: Select a Topic - 10/19
 - Milestone 2: Literature Survey - 10/29
 - Milestone 3: Design Document - 11/5
 - Milestone 4: Final Presentation - 12/14

<https://gwdistsys20.github.io/project/>

WHICH LB ARCHITECTURE IS BETTER?

