Computing While Charging

Building a Distributed Computing Infrastructure Using Smartphones

Mustafa Y. Arslan - Indrajeet Singh - Shailendra Singh

Harsha V. Madhyastha - Karthikeyan Sundaresan - Srikanth V. Krishnamurthy

Smartphones and Computing

- Smartphones with higher CPU clock speeds, more CPU cores, and so on.
  - real computers in our pockets
- Enterprises are also adopting smartphones.
- **Problem:** The real computing power of smartphones is yet to be tapped into.
  - battery drains quickly -- long idle charging times (e.g., at night)

Idle Phones Put Back to Work

- Utilize idle charging periods for executing distributed jobs on phones, for an enterprise.
- 100s-1000s of smartphones, working ***in parallel***

Idle Phones Put Back to Work

- Utilize idle charging periods for executing distributed jobs on phones, for an enterprise.
- 100s-1000s of smartphones, working ***in parallel***
- Why would I bother? I have a datacenter full of PCs.

Idle Phones Put Back to Work

- Utilize idle charging periods for executing distributed jobs on phones, for an enterprise.
- 100s-1000s of smartphones, working ***in parallel***
- Why would I bother? I have a datacenter full of PCs.
  - untapped horsepower -- phones will run the same code as your servers

Idle Phones Put Back to Work

- Utilize idle charging periods for executing distributed jobs on phones, for an enterprise.
- 100s-1000s of smartphones, working ***in parallel***
- Why would I bother? I have a datacenter full of PCs.
  - untapped horsepower -- phones will run the same code as your servers
  - no cost to bootstrap (phones are already in hand)

Idle Phones Put Back to Work

- Utilize idle charging periods for executing distributed jobs on phones, for an enterprise.
- 100s-1000s of smartphones, working ***in parallel***
- Why would I bother? I have a datacenter full of PCs.
  - untapped horsepower -- phones will run the same code as your servers
  - no cost to bootstrap (phones are already in hand)
  - a phone consumes ~5% of the energy of a PC.

Idle Phones Put Back to Work

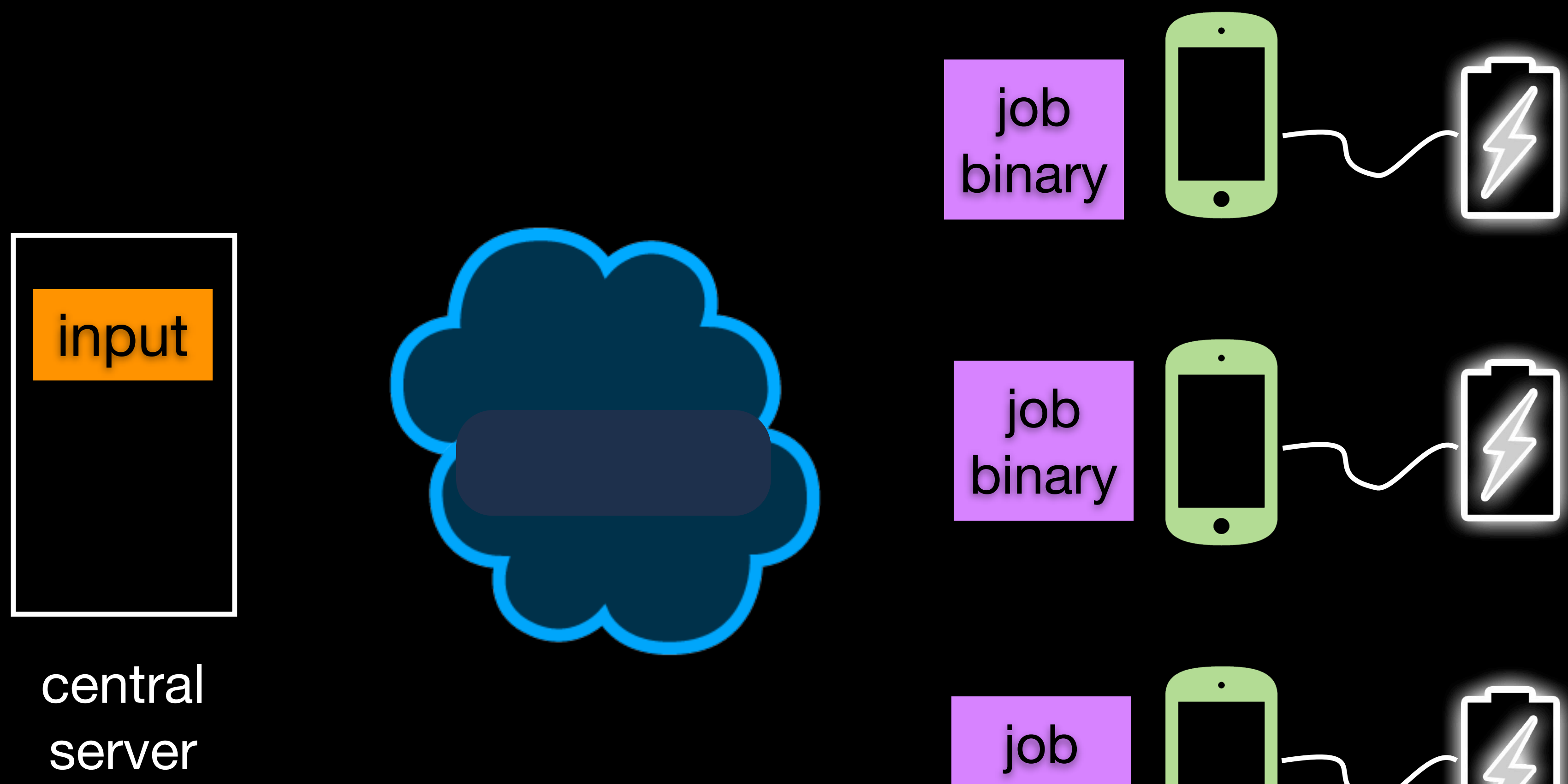
- Utilize idle charging periods for executing distributed jobs on phones, for an enterprise.
- 100s-1000s of smartphones, working ***in parallel***
- Why would I bother? I have a datacenter full of PCs.
  - untapped horsepower -- phones will run the same code as your servers
  - no cost to bootstrap (phones are already in hand)
  - a phone consumes ~5% of the energy of a PC.
  - no wiring, switching or cooling needed.

Idle Phones Put Back to Work

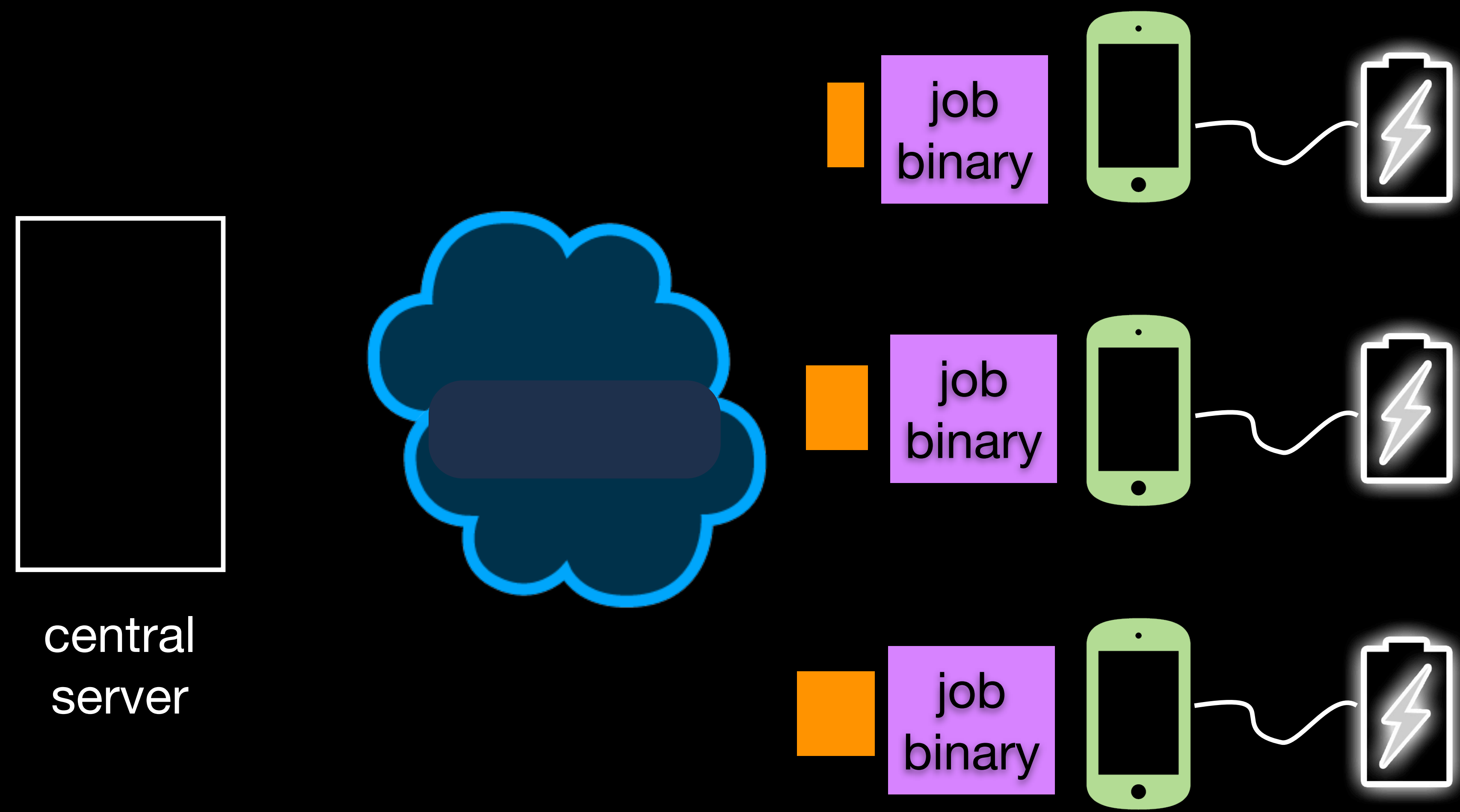
- Utilize idle charging periods for executing distributed jobs on phones, for an enterprise.
- 100s-1000s of smartphones, working ***in parallel***
- Why would I bother? I have a datacenter full of PCs.
    - untapped horsepower -- phones will run the same code as your servers
    - no cost to bootstrap (phones are already in hand)
    - a phone consumes ~5% of the energy of a PC.
    - no wiring, switching or cooling needed.
- Case to consider smartphones as a supplement for existing computational systems.

The Desired System

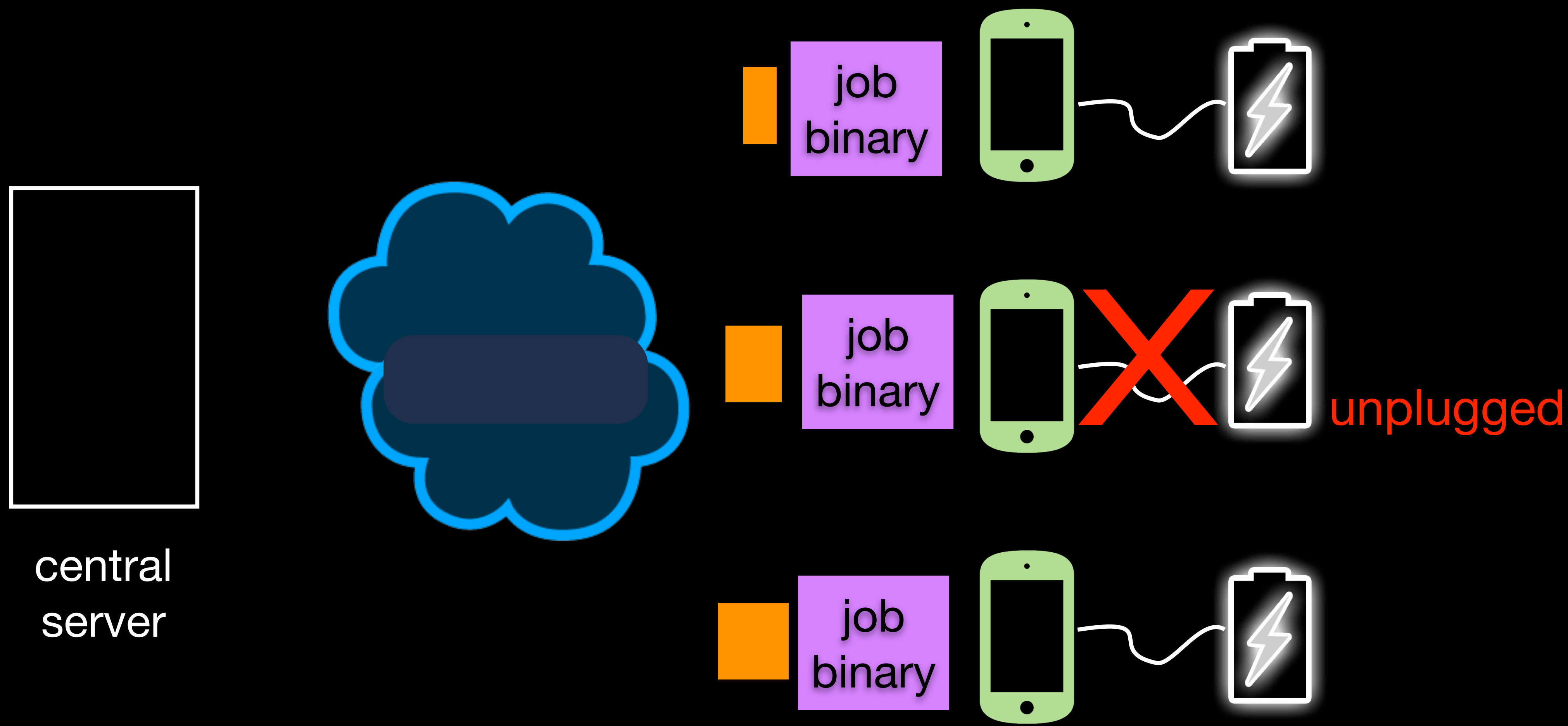
The Desired System



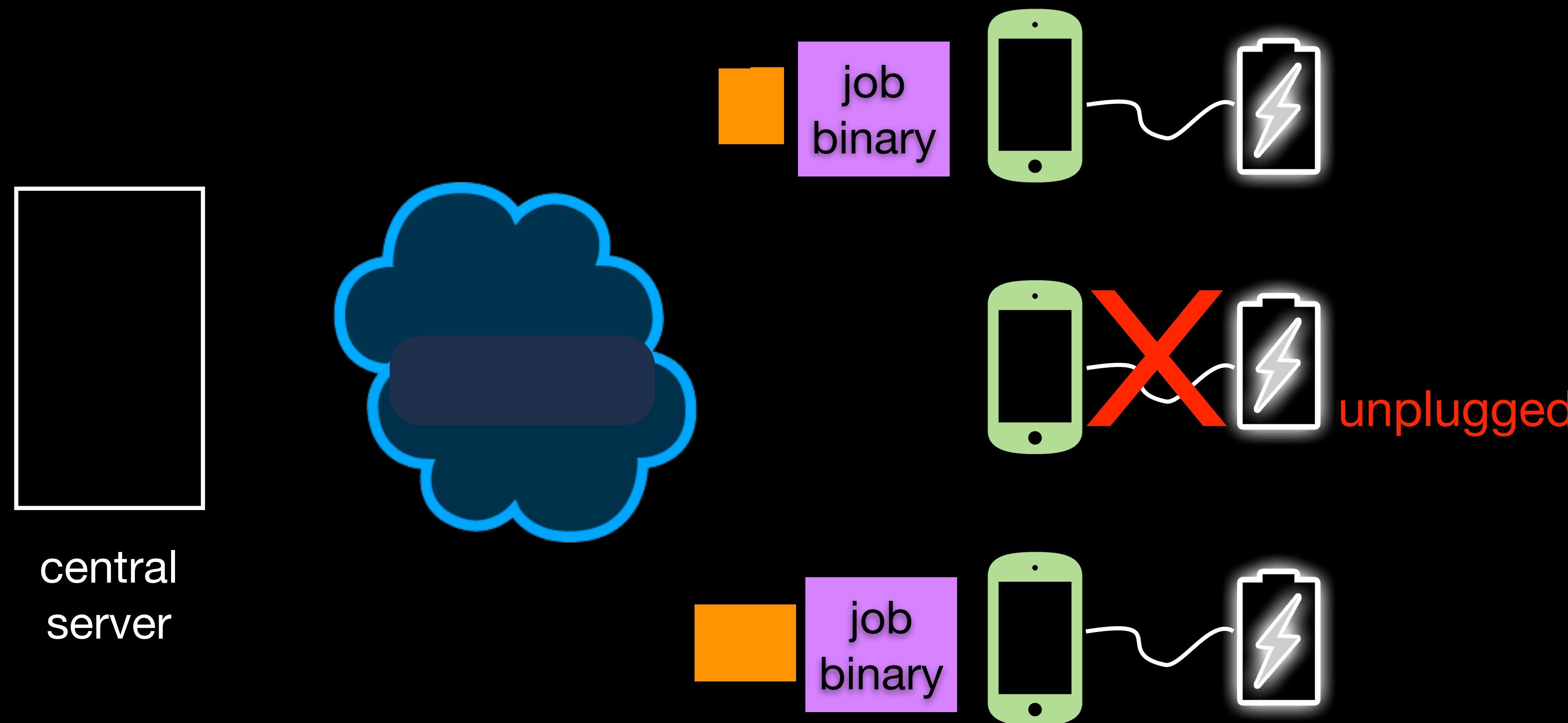
The Desired System



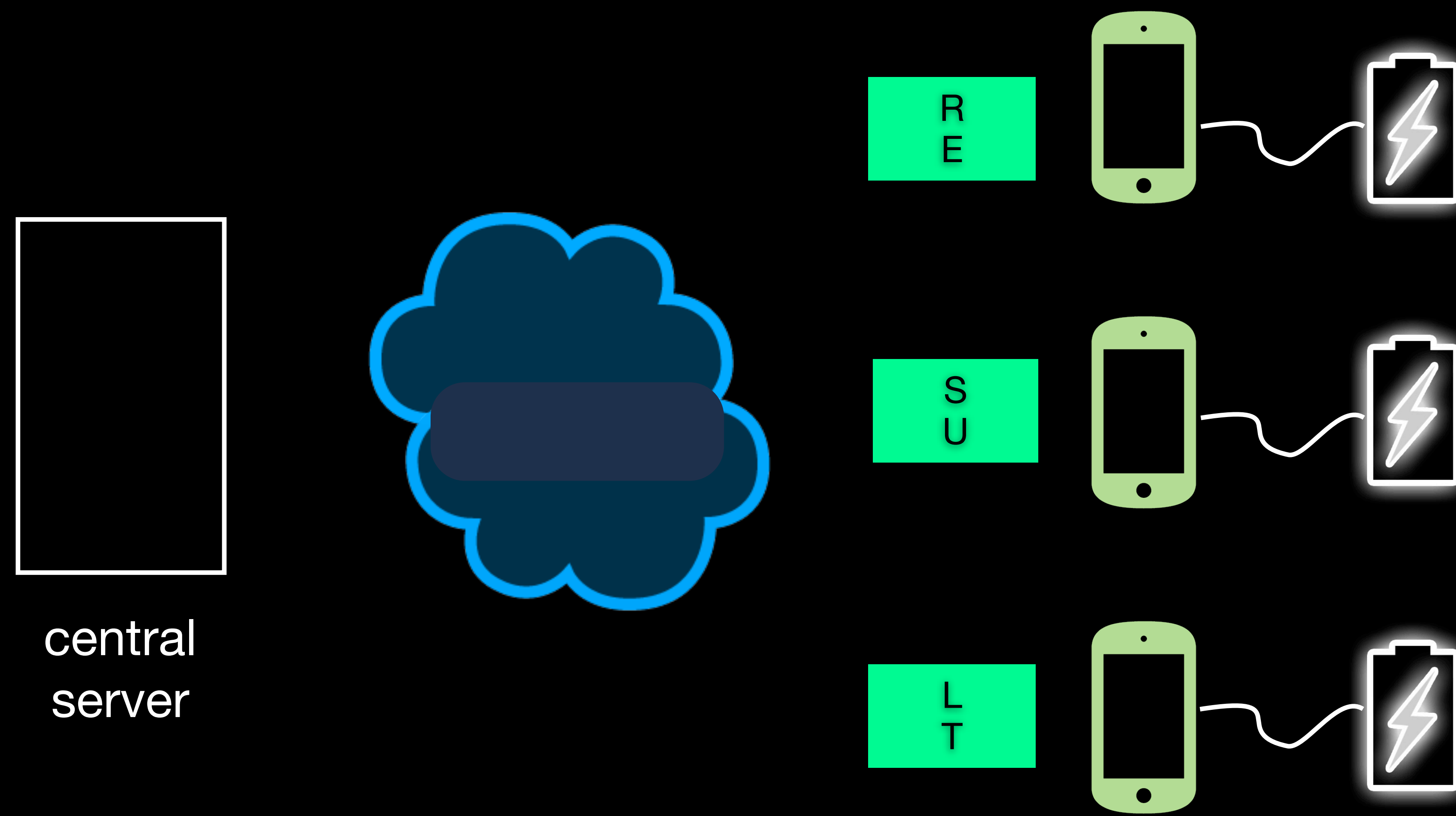
The Desired System



The Desired System



The Desired System



The Desired System

Our Contribution

Our Contribution

- Design & implement *CWC*.

Our Contribution

- Design & implement *CWC*.
- What is not novel: utilizing idle CPUs (e.g., *Condor: A Hunter of Idle Workstations [Litzkow, Livny, Mutka]*).

Our Contribution

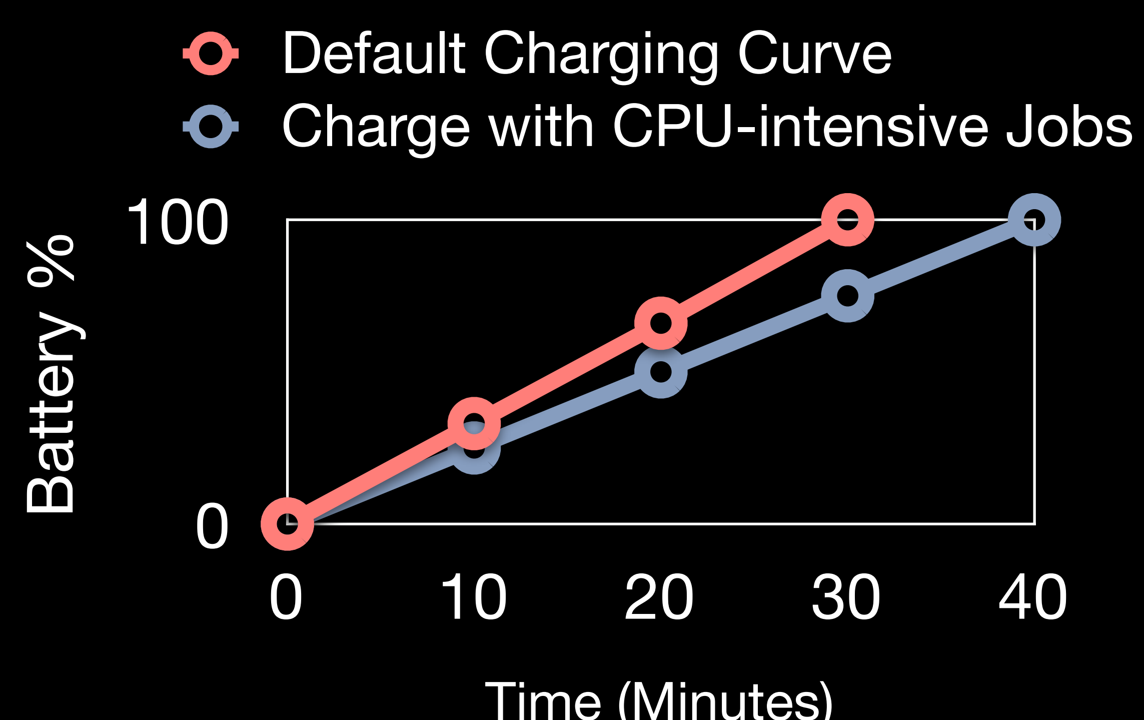
- Design & implement *CWC*.
- What is not novel: utilizing idle CPUs (e.g., *Condor: A Hunter of Idle Workstations [Litzkow, Livny, Mutka]*).
- What is novel: algorithm to optimally distribute computation across smartphones with non-uniform bandwidths

Our Contribution

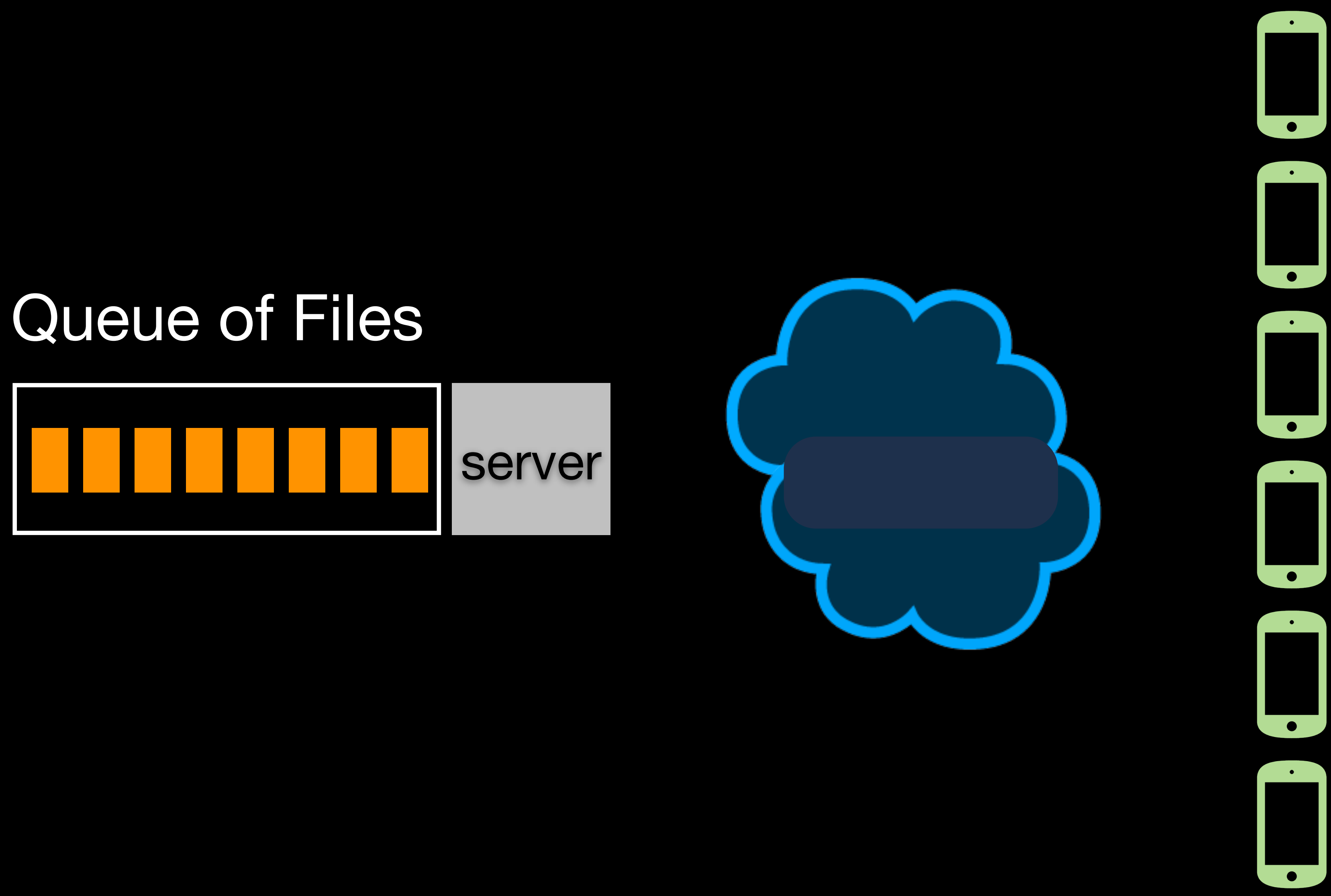
- Design & implement *CWC*.
- What is not novel: utilizing idle CPUs (e.g., *Condor: A Hunter of Idle Workstations [Litzkow, Livny, Mutka]*).
- What is novel: algorithm to optimally distribute computation across smartphones with non-uniform bandwidths
  - Non-uniform wireless bandwidth calls for novel schedulers (such as *CWC*).

Our Contribution

- Design & implement *CWC*.
- What is not novel: utilizing idle CPUs (e.g., *Condor: A Hunter of Idle Workstations [Litzkow, Livny, Mutka]*).
- What is novel: algorithm to optimally distribute computation across smartphones with non-uniform bandwidths
  - Non-uniform wireless bandwidth calls for novel schedulers (such as *CWC*).
  - Unique challenges not previously addressed.



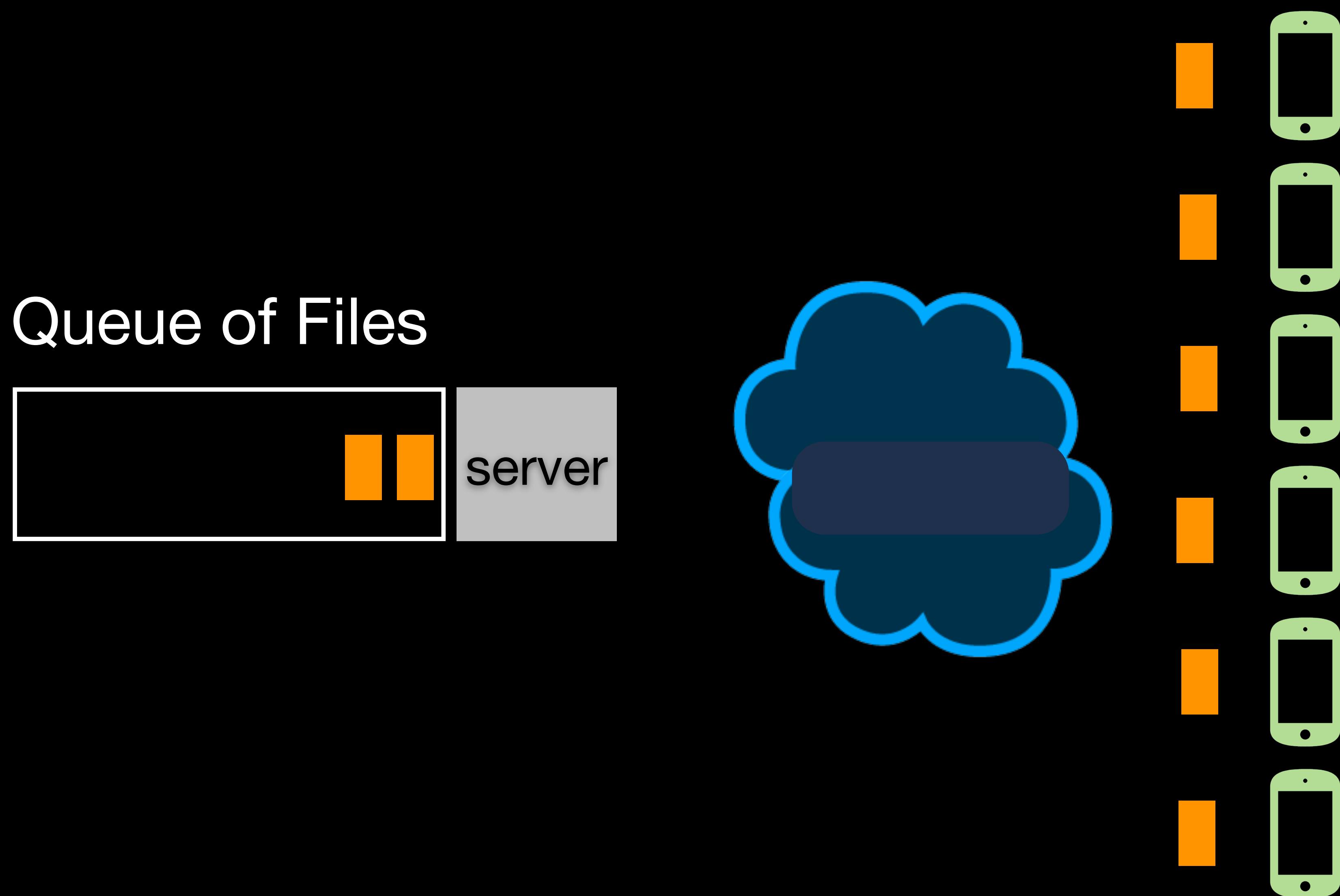
Effect of Bandwidth on Scheduling



Phones have identical CPUs but varying bandwidths

- Files to be processed by the phones (queue a file until next phone becomes available).
- The server logs the service time (queueing + transfer + processing) of each file.

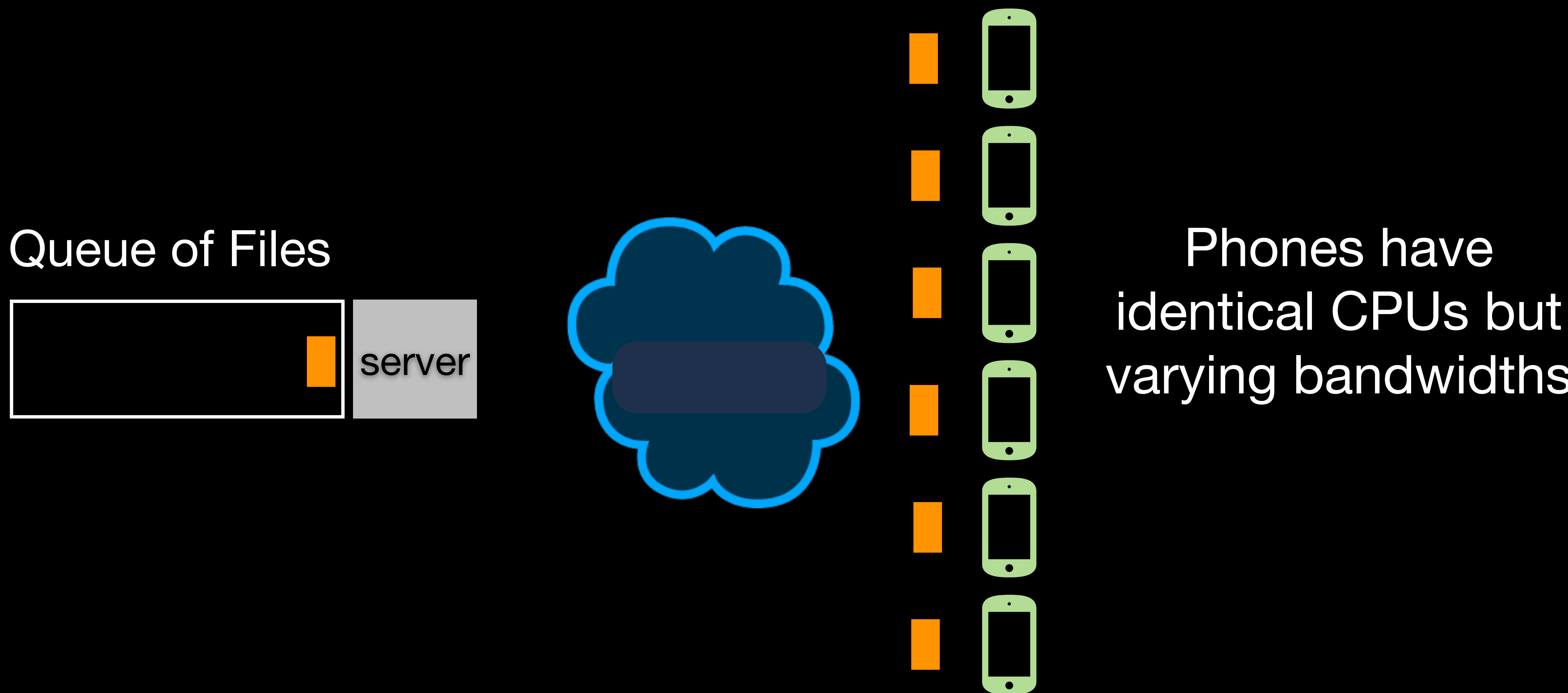
Effect of Bandwidth on Scheduling



Phones have identical CPUs but varying bandwidths

- Files to be processed by the phones (queue a file until next phone becomes available).
- The server logs the service time (queueing + transfer + processing) of each file.

Effect of Bandwidth on Scheduling



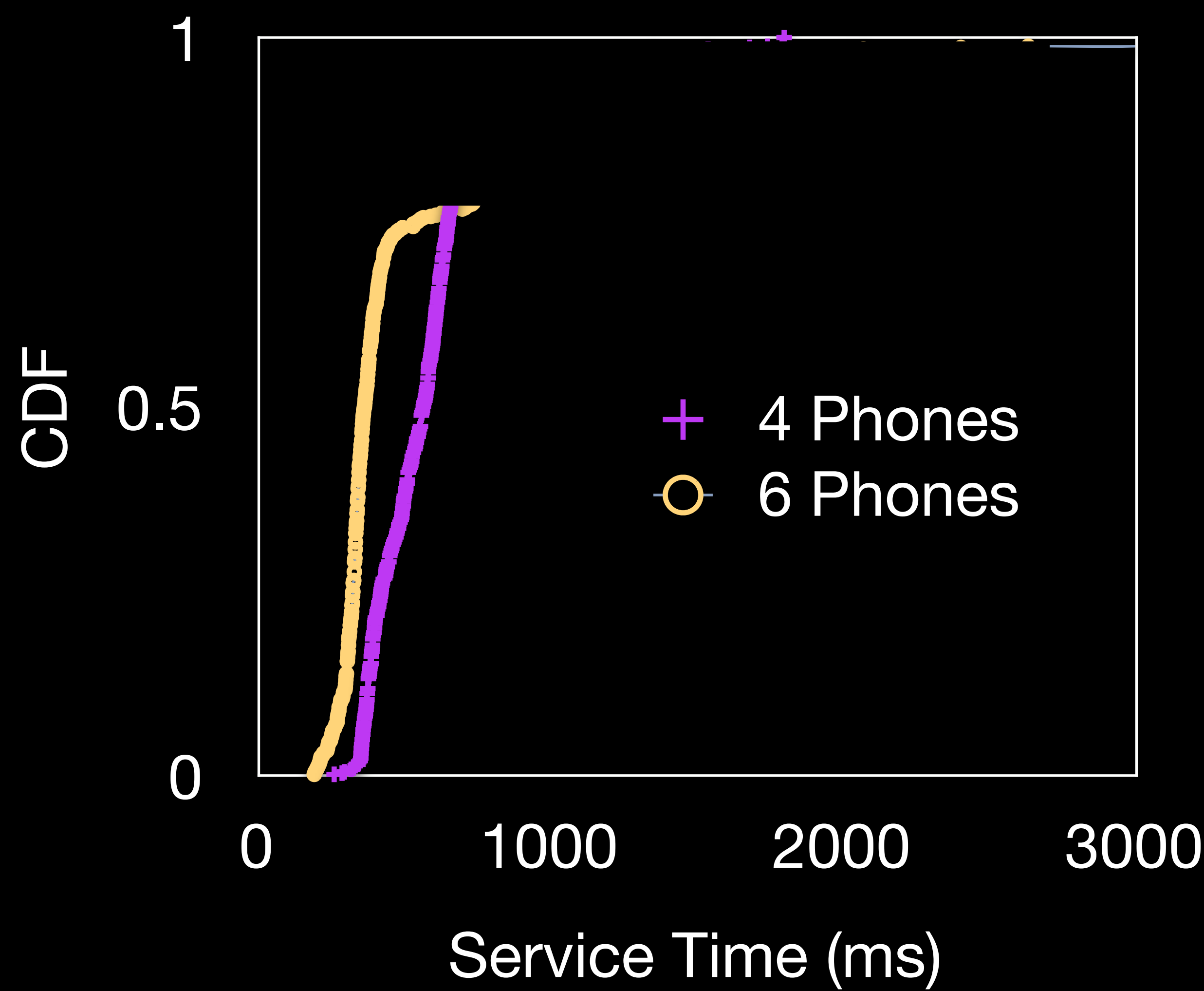
- Files to be processed by the phones (queue a file until next phone becomes available).
- The server logs the service time (queueing + transfer + processing) of each file.

Effect of Bandwidth on Scheduling

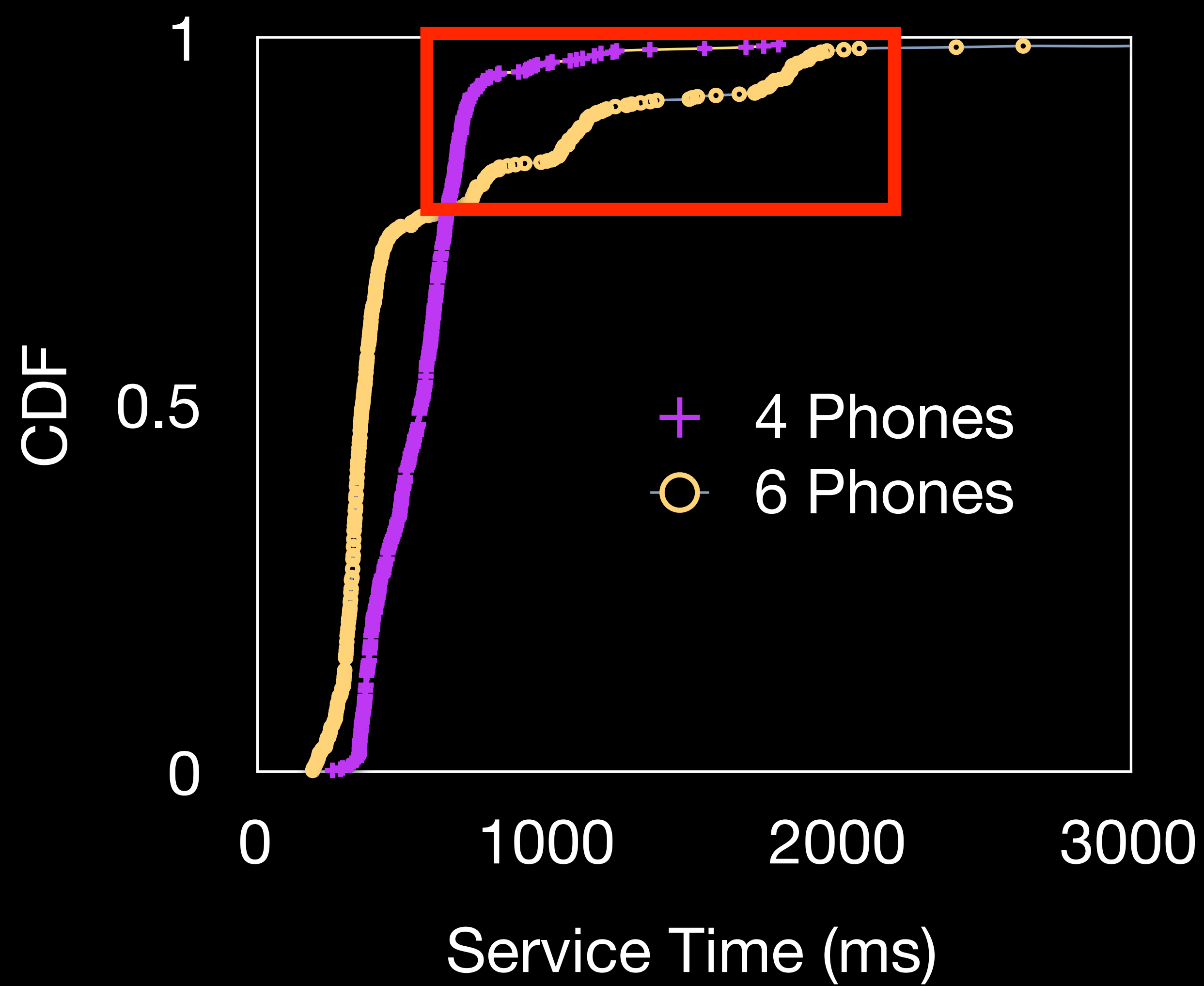
Phones have identical CPUs but varying bandwidths

- Two phones with the lowest bandwidths are removed and the experiment is repeated.

Too Much Parallelism Hurts



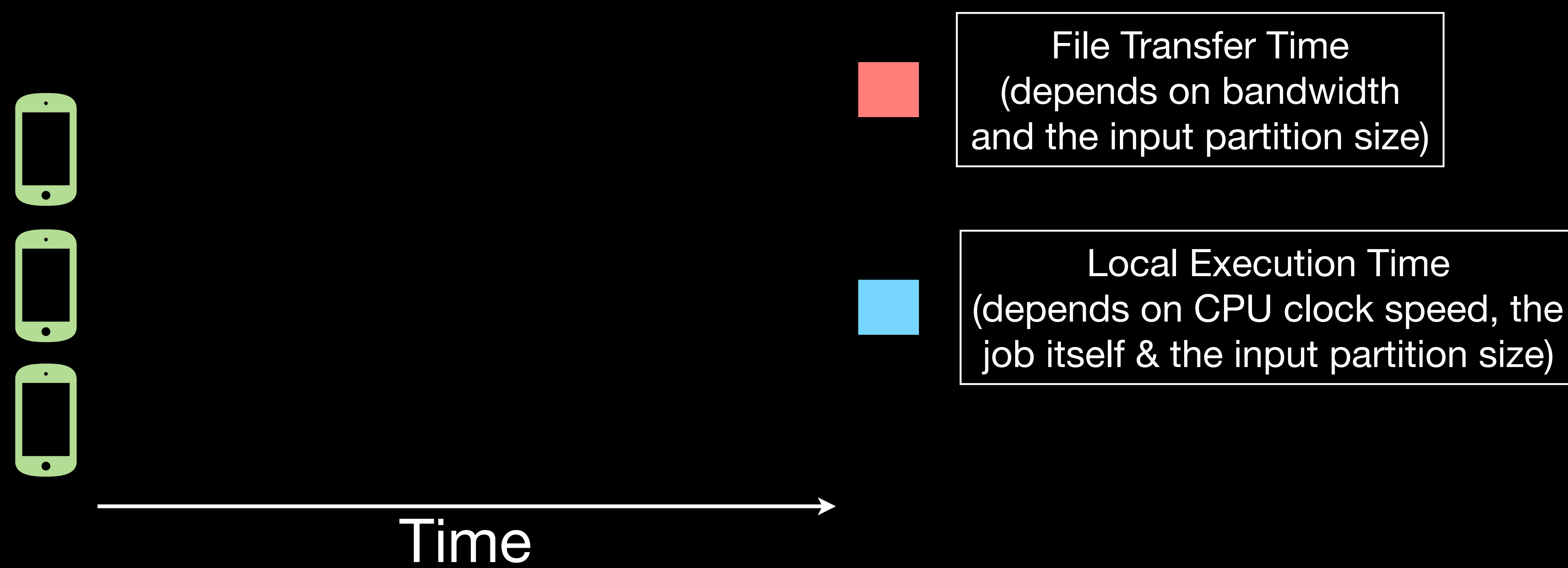
Too Much Parallelism Hurts



- Using only phones with high bandwidth can compensate for reduced number of worker phones.
  - it is not a straight-forward choice to leverage the full parallelism!

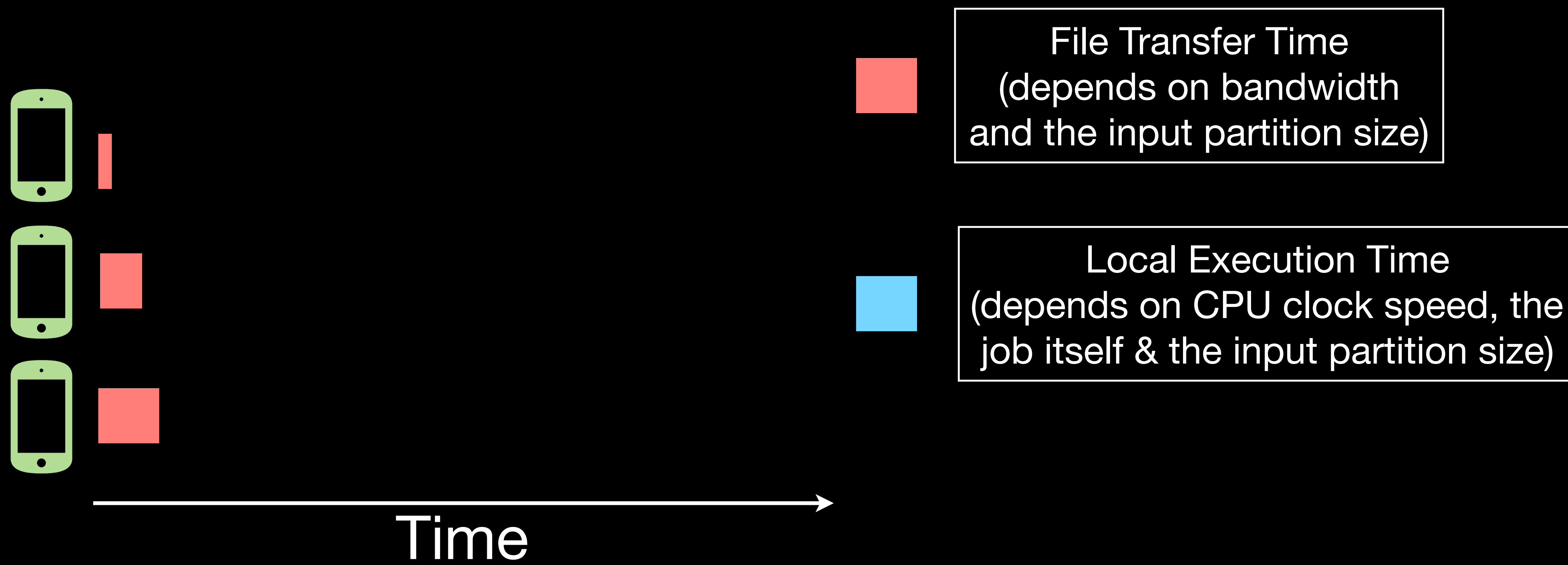
Problem Statement

- Given a set of jobs and their input files, how do we partition and distribute these files across a set of smartphones, ***to minimize the makespan?***



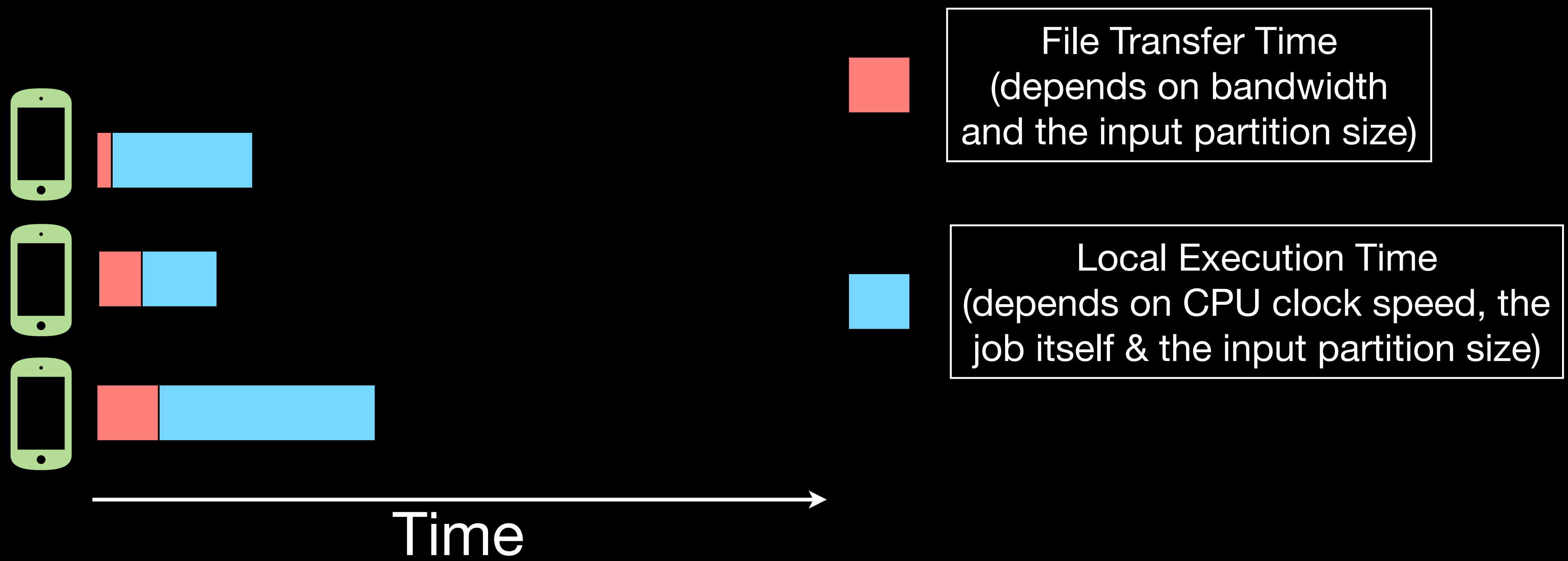
Problem Statement

- Given a set of jobs and their input files, how do we partition and distribute these files across a set of smartphones, ***to minimize the makespan?***



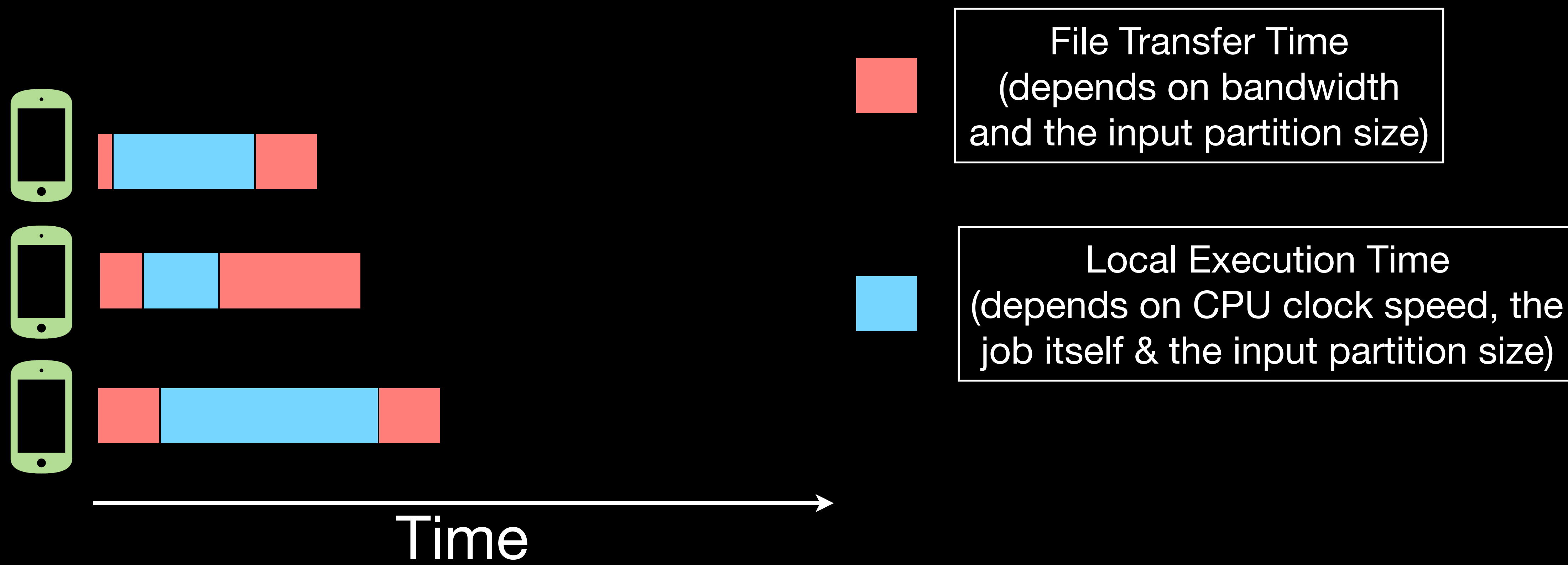
Problem Statement

- Given a set of jobs and their input files, how do we partition and distribute these files across a set of smartphones, ***to minimize the makespan?***



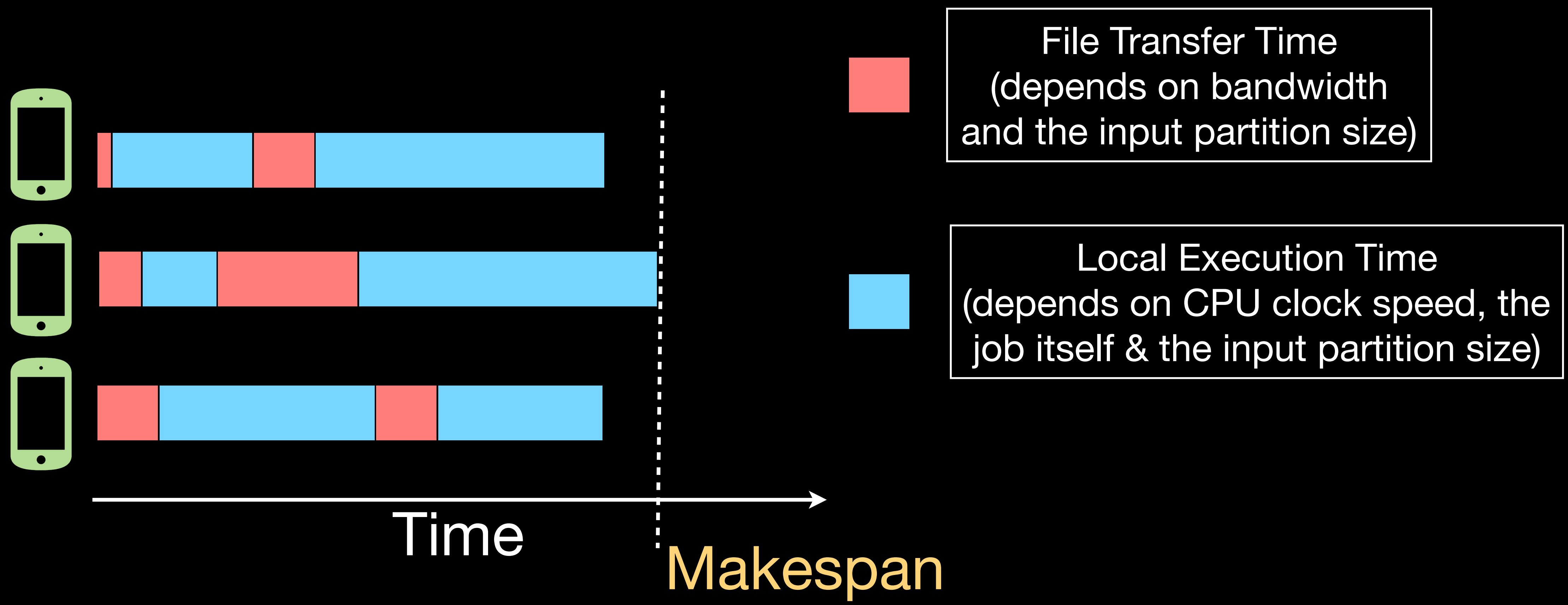
Problem Statement

- Given a set of jobs and their input files, how do we partition and distribute these files across a set of smartphones, ***to minimize the makespan?***



Problem Statement

- Given a set of jobs and their input files, how do we partition and distribute these files across a set of smartphones, ***to minimize the makespan?***



Predicting Execution Times

Predicting Execution Times

- bandwidth to each phone periodically measured. but cannot measure the local execution time for every job-phone pair.

Predicting Execution Times

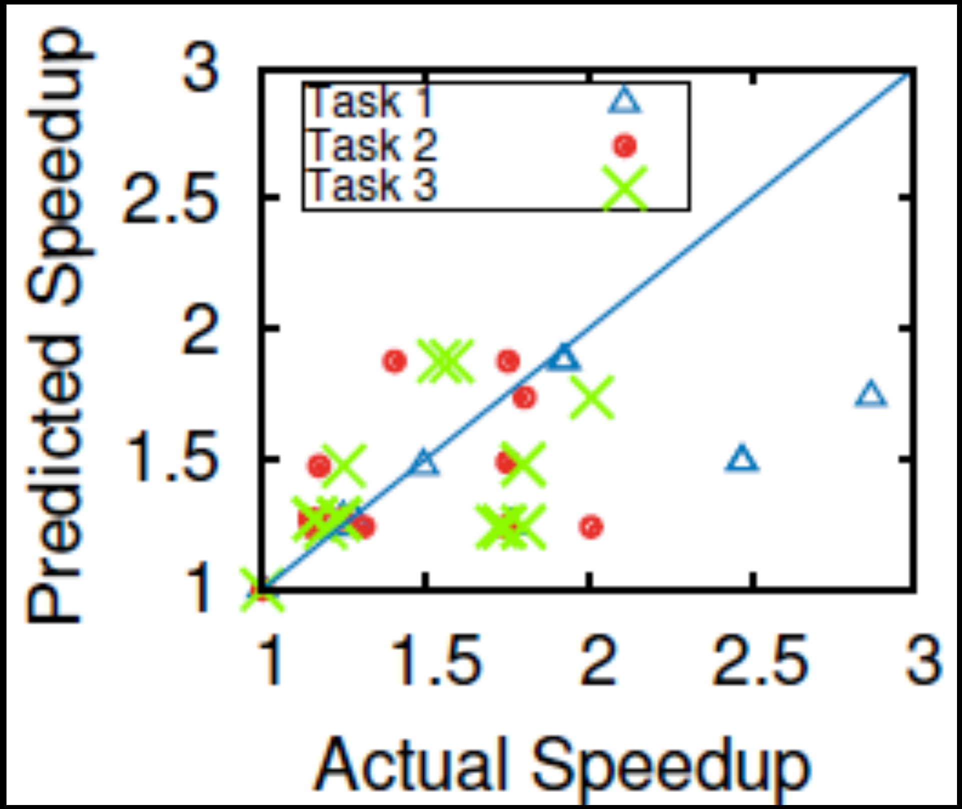
- bandwidth to each phone periodically measured. but cannot measure the local execution time for every job-phone pair.
  - run each job j using the slowest phone in the system: e.g., with **S** MHz CPU

Predicting Execution Times

- bandwidth to each phone periodically measured. but cannot measure the local execution time for every job-phone pair.
  - run each job j using the slowest phone in the system: e.g., with **S** MHz CPU
  - if the slowest phone takes $\mathbf{T_s}$ ms, then another phone with **A** MHz CPU should take $\mathbf{T_s * S / A}$ ms.

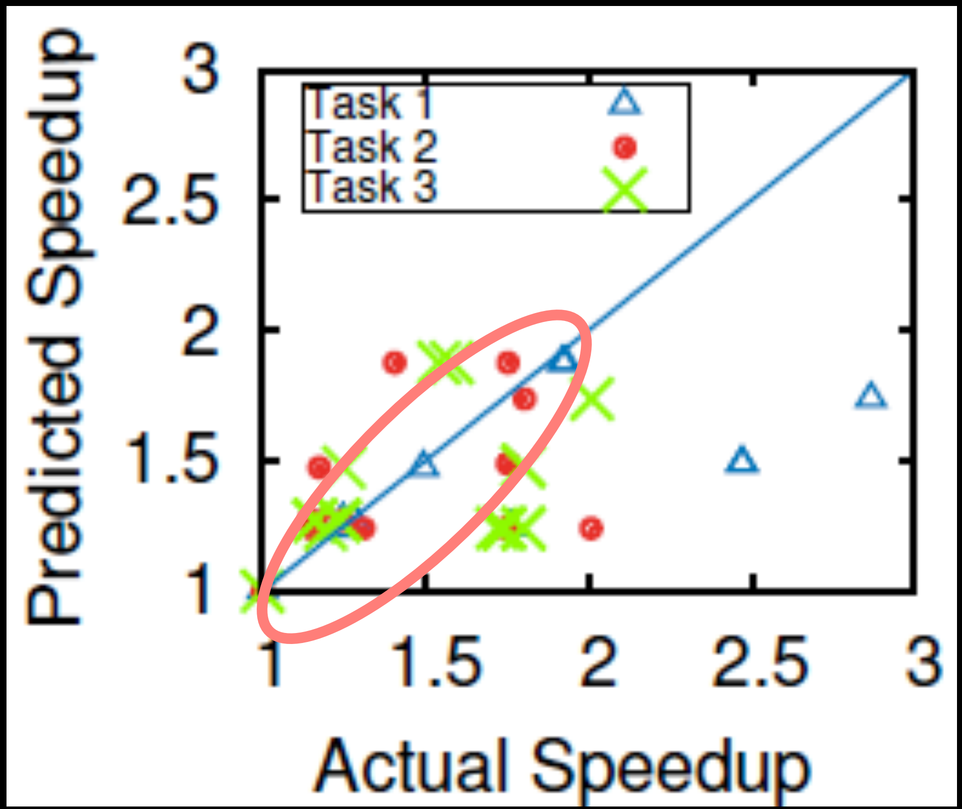
Predicting Execution Times

- bandwidth to each phone periodically measured. but cannot measure the local execution time for every job-phone pair.
  - run each job j using the slowest phone in the system: e.g., with **S** MHz CPU
  - if the slowest phone takes $\mathbf{T_s}$ ms, then another phone with **A** MHz CPU should take $\mathbf{T_s * S / A}$ ms.



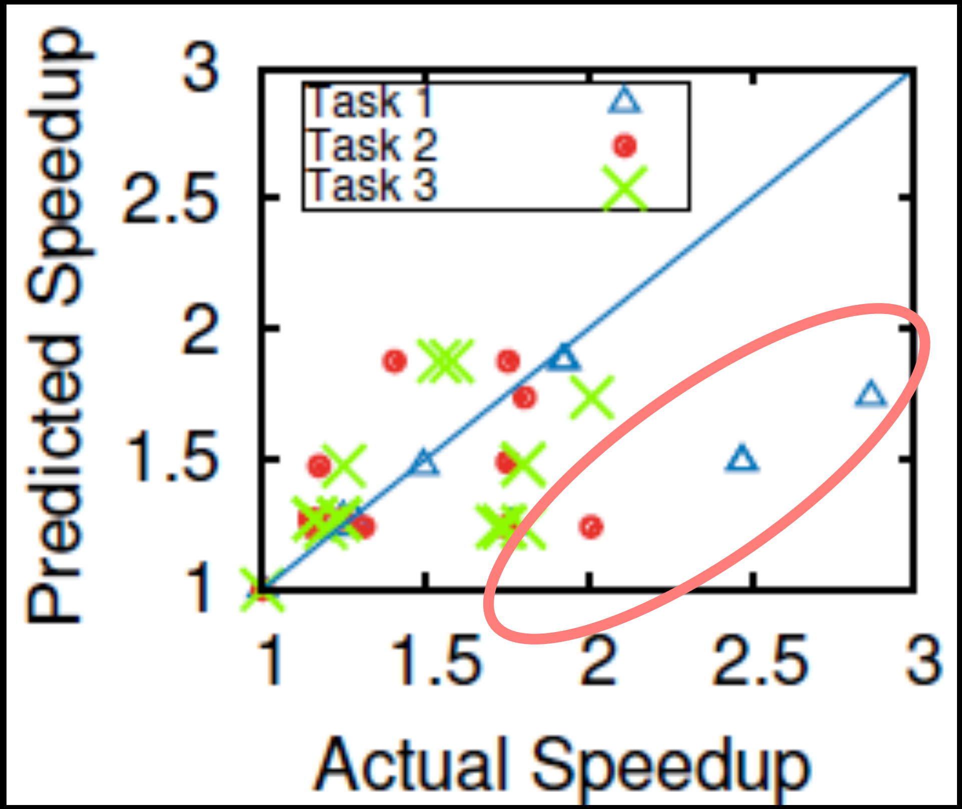
Predicting Execution Times

- bandwidth to each phone periodically measured. but cannot measure the local execution time for every job-phone pair.
  - run each job j using the slowest phone in the system: e.g., with **S** MHz CPU
  - if the slowest phone takes $\mathbf{T_s}$ ms, then another phone with **A** MHz CPU should take $\mathbf{T_s * S / A}$ ms.



Predicting Execution Times

- bandwidth to each phone periodically measured. but cannot measure the local execution time for every job-phone pair.
  - run each job j using the slowest phone in the system: e.g., with **S** MHz CPU
  - if the slowest phone takes $\mathbf{T_s}$ ms, then another phone with **A** MHz CPU should take $\mathbf{T_s * S / A}$ ms.



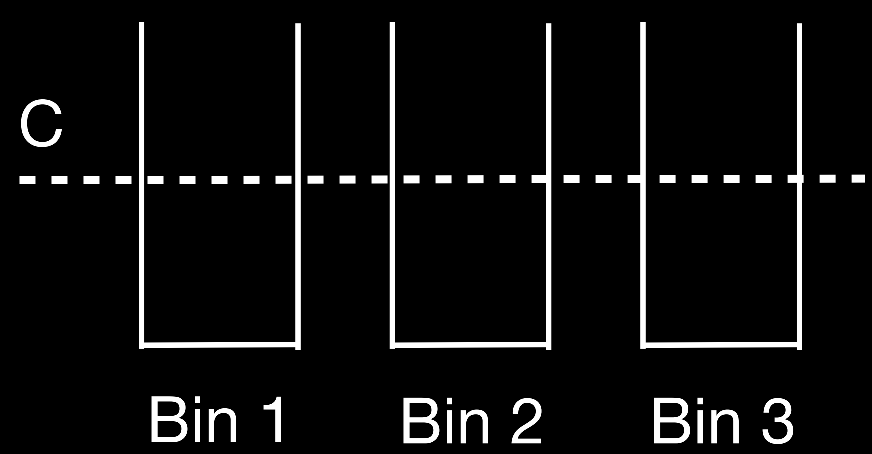
wrong estimates are corrected using execution reports sent to the central server.

Minimum Height Bin Packing

- Given a finite set of items *U*, a size for each item in *U* and a bin capacity *C*.
  - partition *U* into disjoint sets U_1, U_2, .., U_n s.t. the sum of the item sizes in each U_i is <= *C*.

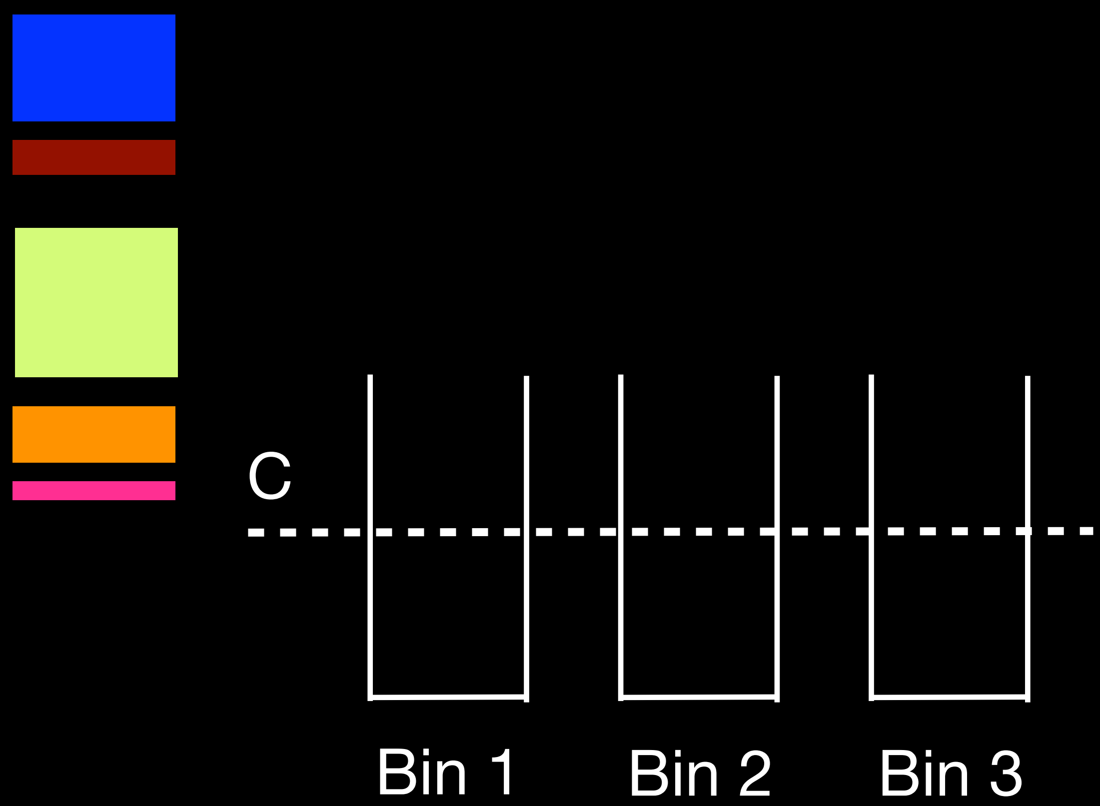
Minimum Height Bin Packing

- Given a finite set of items *U*, a size for each item in *U* and a bin capacity *C*.
  - partition *U* into disjoint sets U_1, U_2, .., U_n s.t. the sum of the item sizes in each U_i is <= *C*.



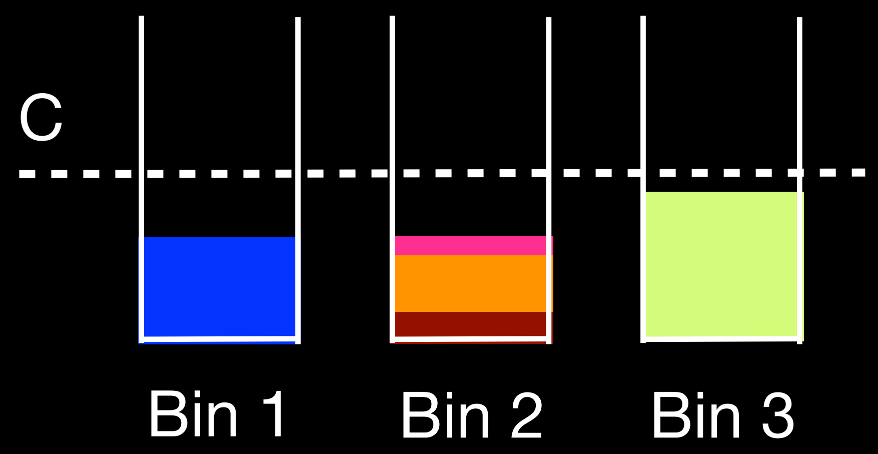
Minimum Height Bin Packing

- Given a finite set of items *U*, a size for each item in *U* and a bin capacity *C*.
  - partition *U* into disjoint sets U_1, U_2, .., U_n s.t. the sum of the item sizes in each U_i is <= *C*.

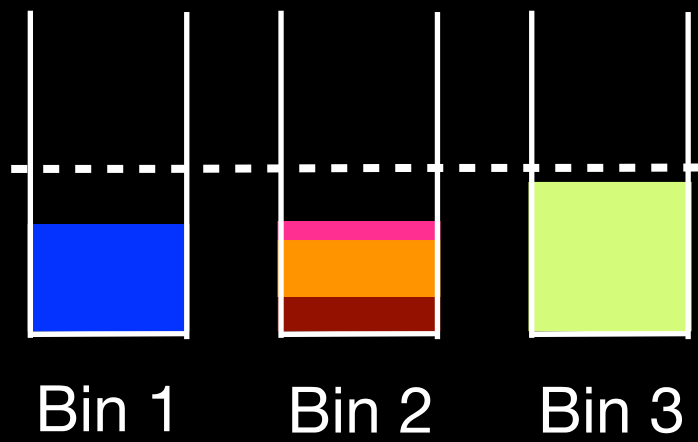


Minimum Height Bin Packing

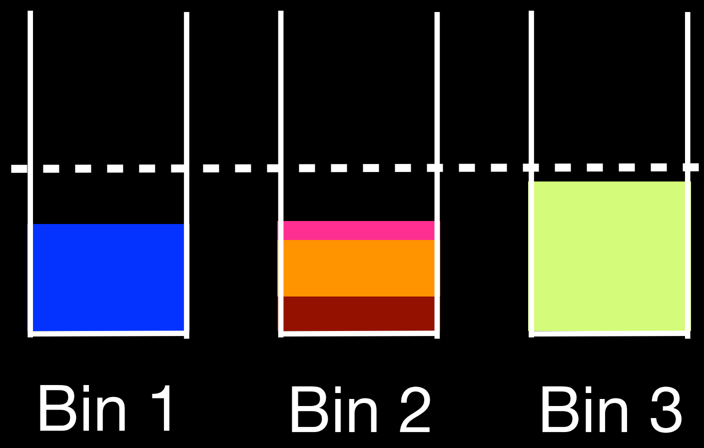
- Given a finite set of items U, a size for each item in U and a bin capacity C.
  - partition U into disjoint sets U_1, U_2, .., U_n s.t. the sum of the item sizes in each U_i is <= C.



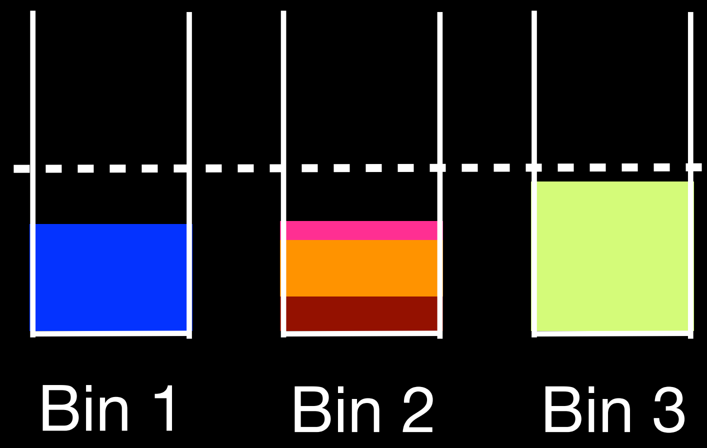
Minimum Height Bin Packing



Minimum Height Bin Packing

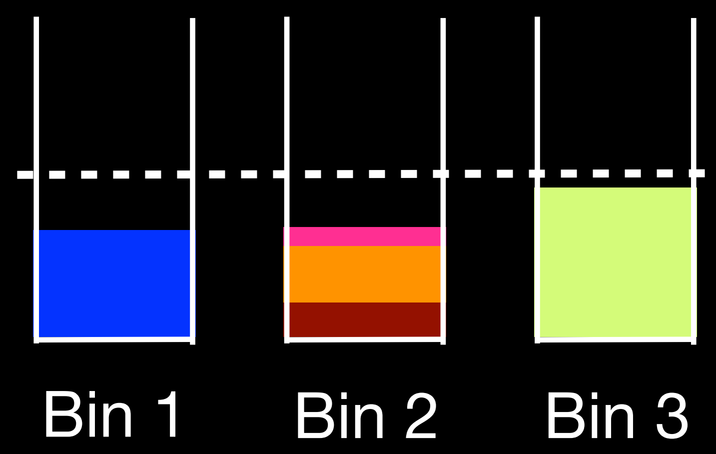


Minimum Height Bin Packing



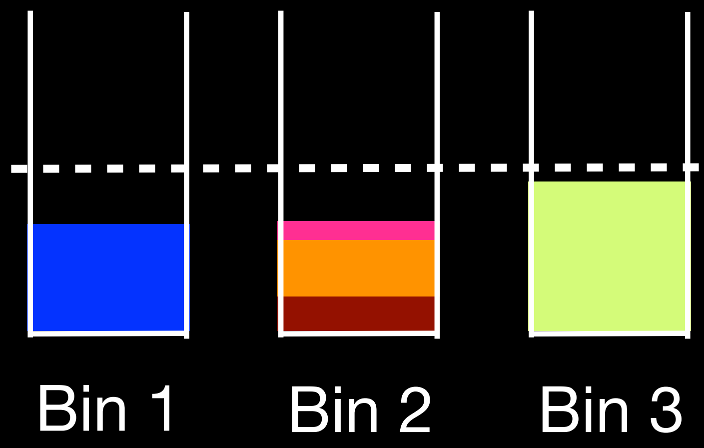
- Each job input is an item (not rigid, i.e., can be partitioned & packed in different bins)

Minimum Height Bin Packing

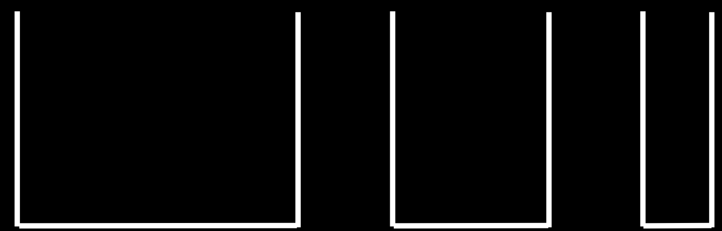


- Each job input is an item (not rigid, i.e., can be partitioned & packed in different bins)
- Cost of partitioning (transferring files adds height depending on phone bandwidth)

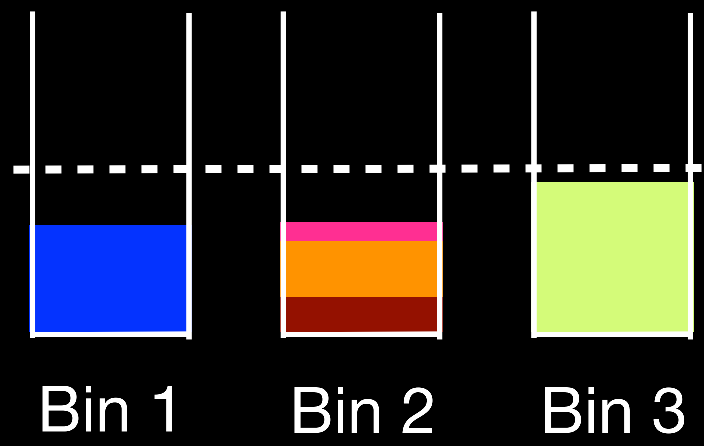
Minimum Height Bin Packing



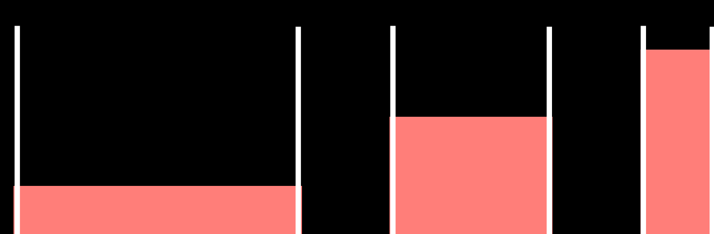
- Each job input is an item (not rigid, i.e., can be partitioned & packed in different bins)
- Cost of partitioning (transferring files adds height depending on phone bandwidth)
- Phones are bins (but they are not identical)



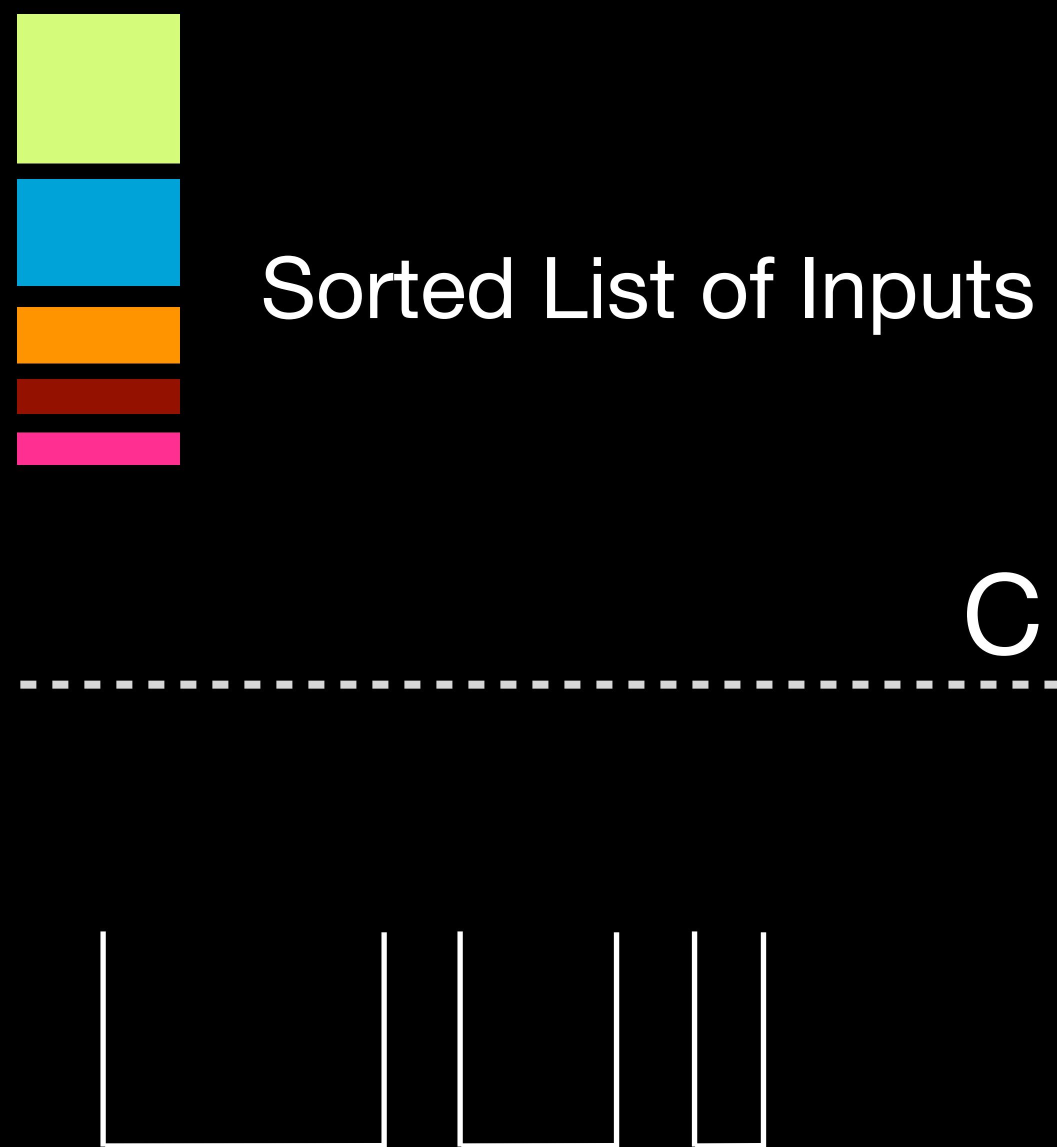
Minimum Height Bin Packing



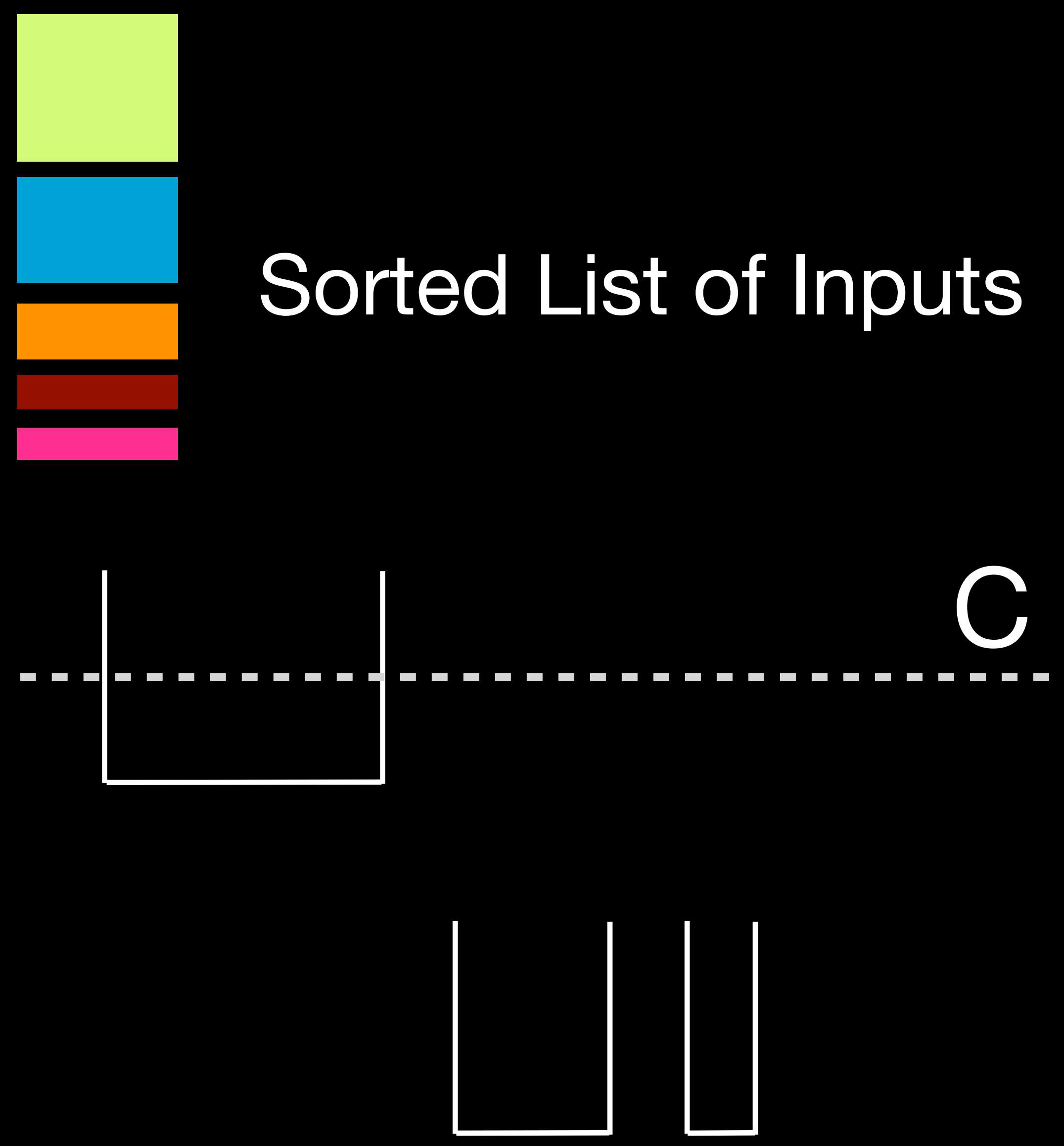
- Each job input is an item (not rigid, i.e., can be partitioned & packed in different bins)
- Cost of partitioning (transferring files adds height depending on phone bandwidth)
- Phones are bins (but they are not identical)
- Items occupy different heights depending on the bin they are packed in.
  - e.g., items behave like liquids.



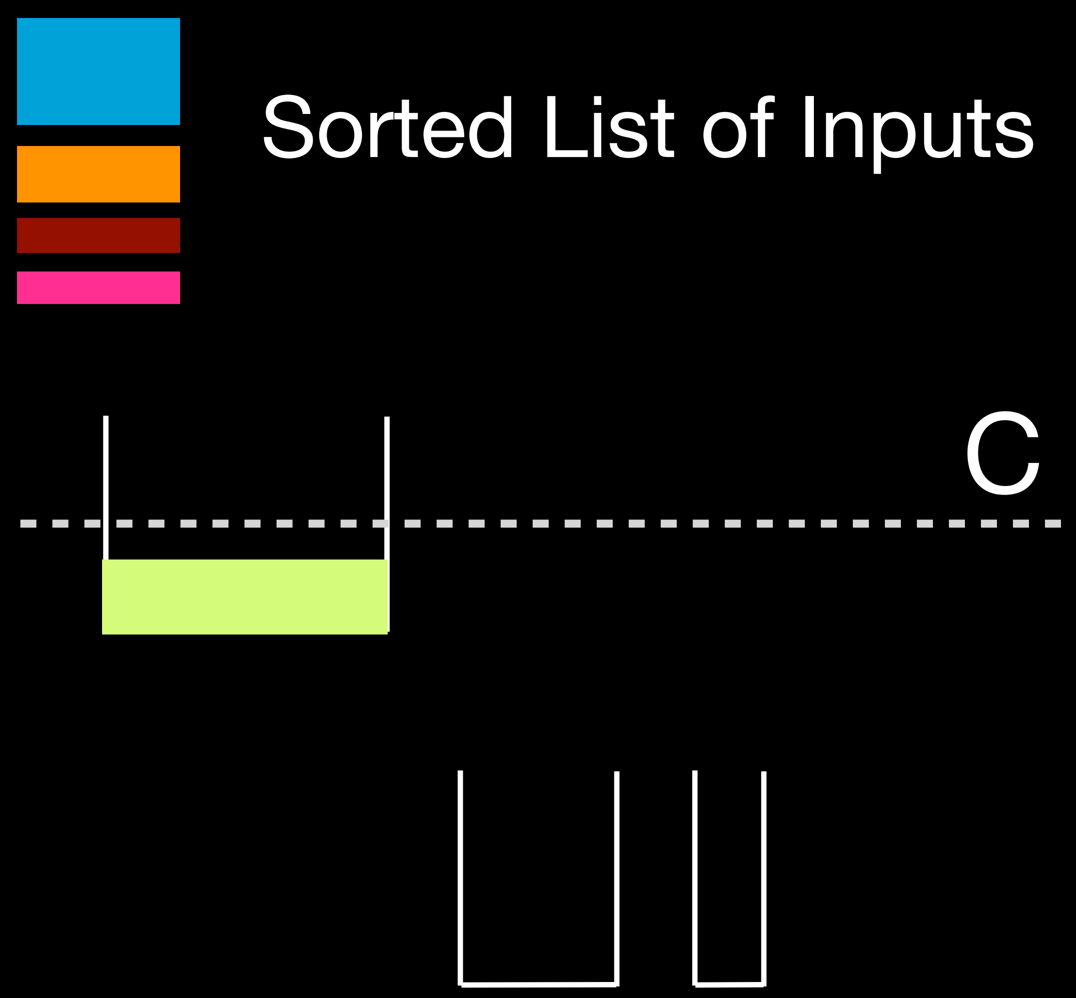
Scheduling Jobs



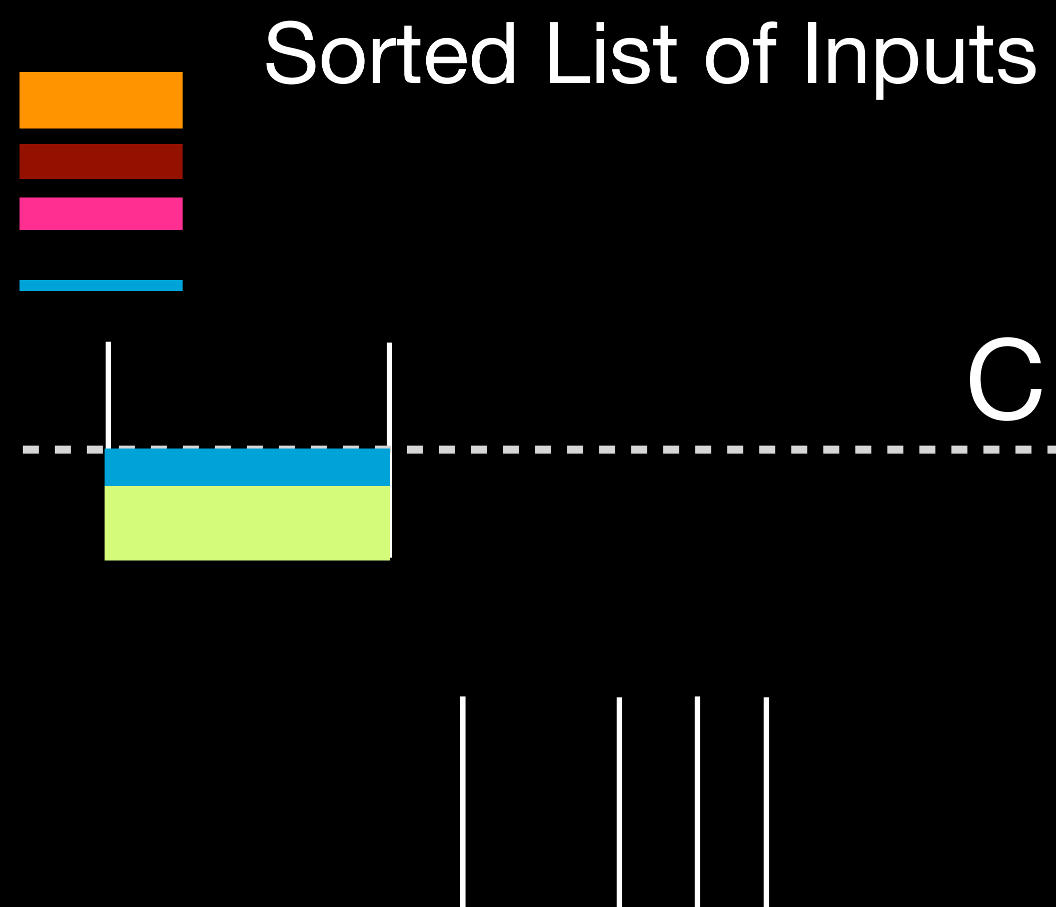
Scheduling Jobs



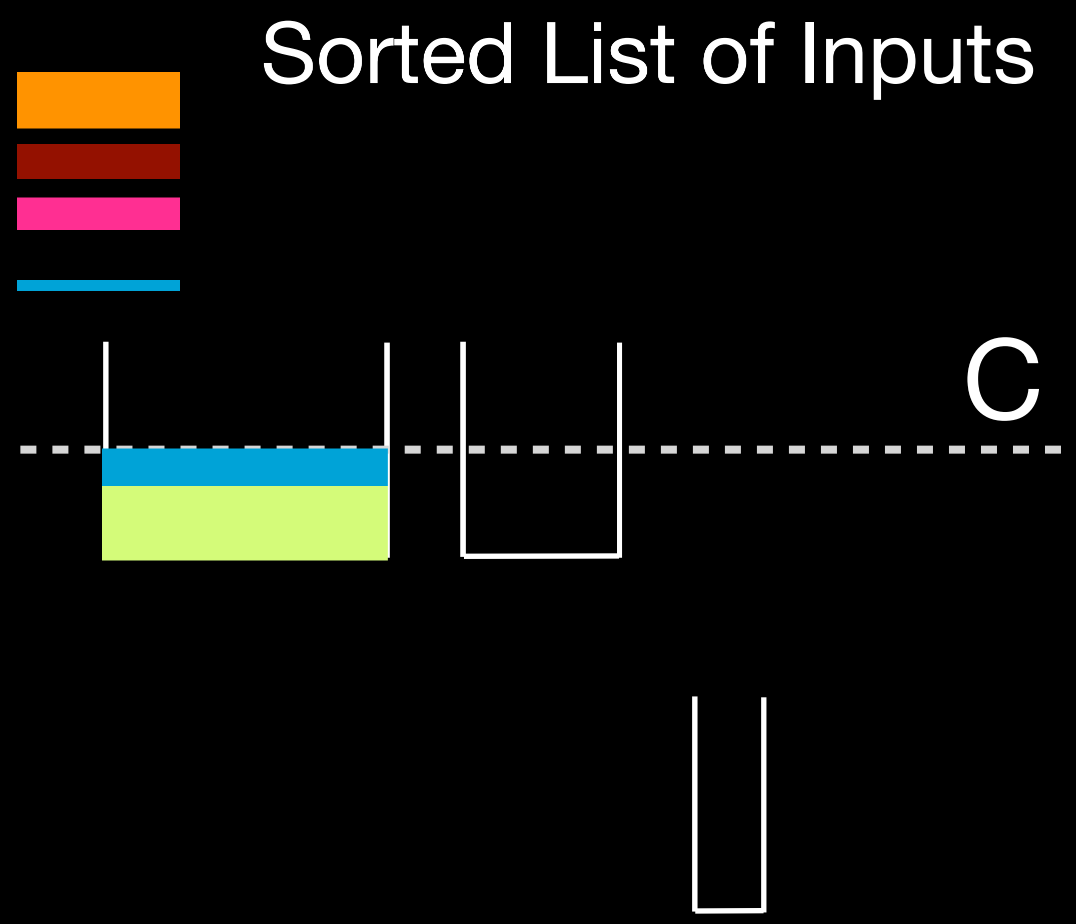
Scheduling Jobs



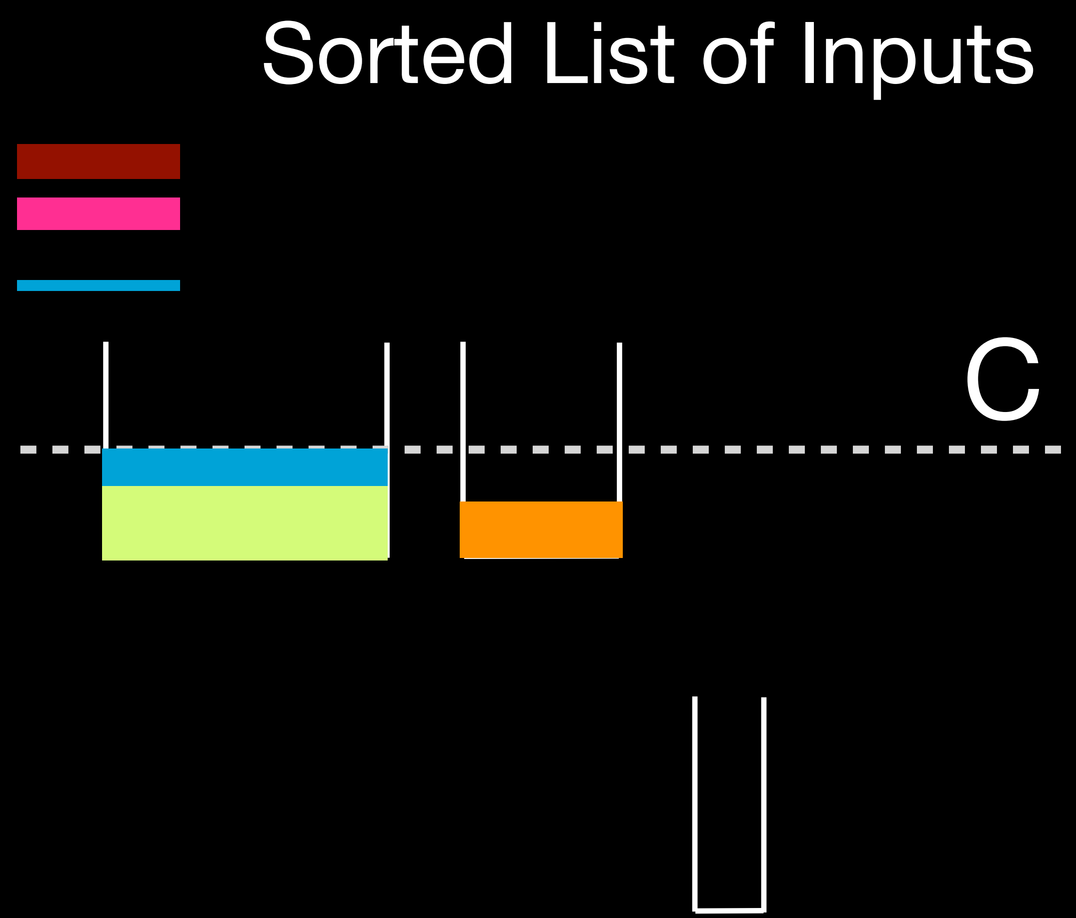
Scheduling Jobs



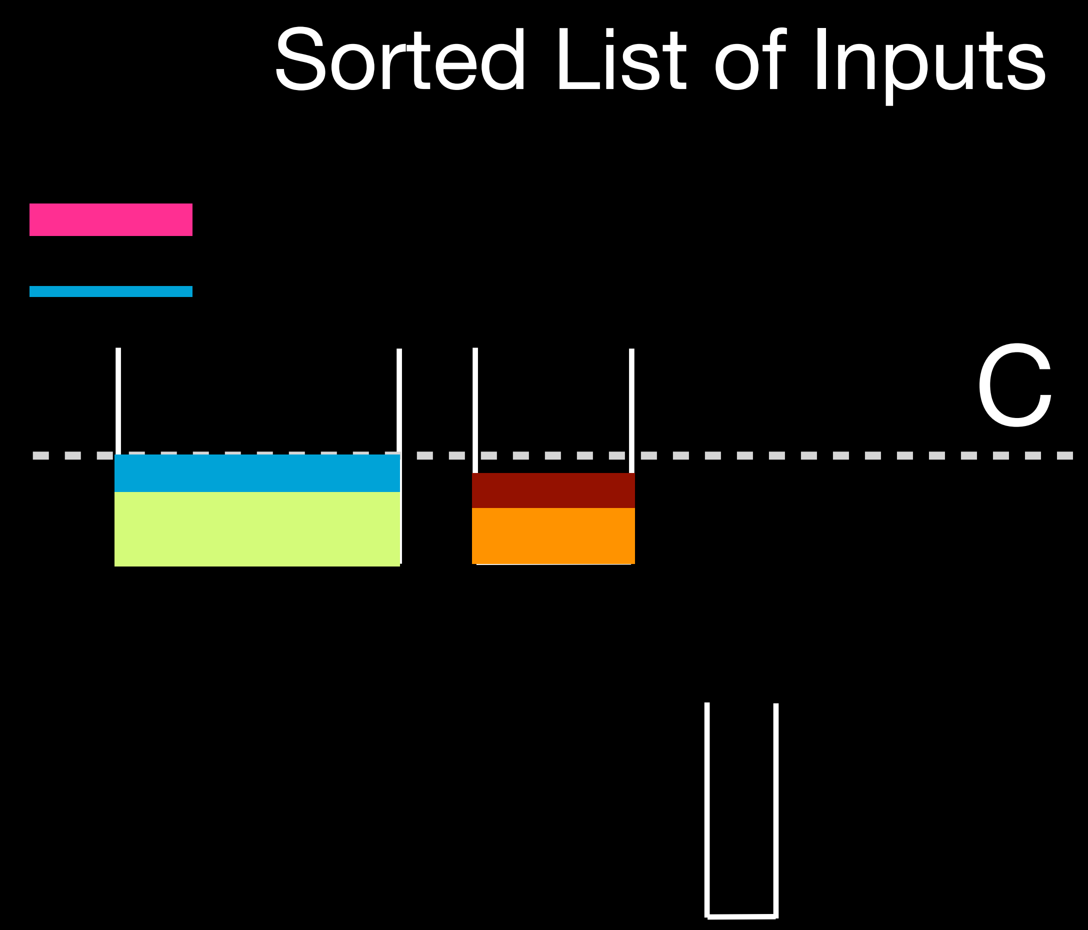
Scheduling Jobs



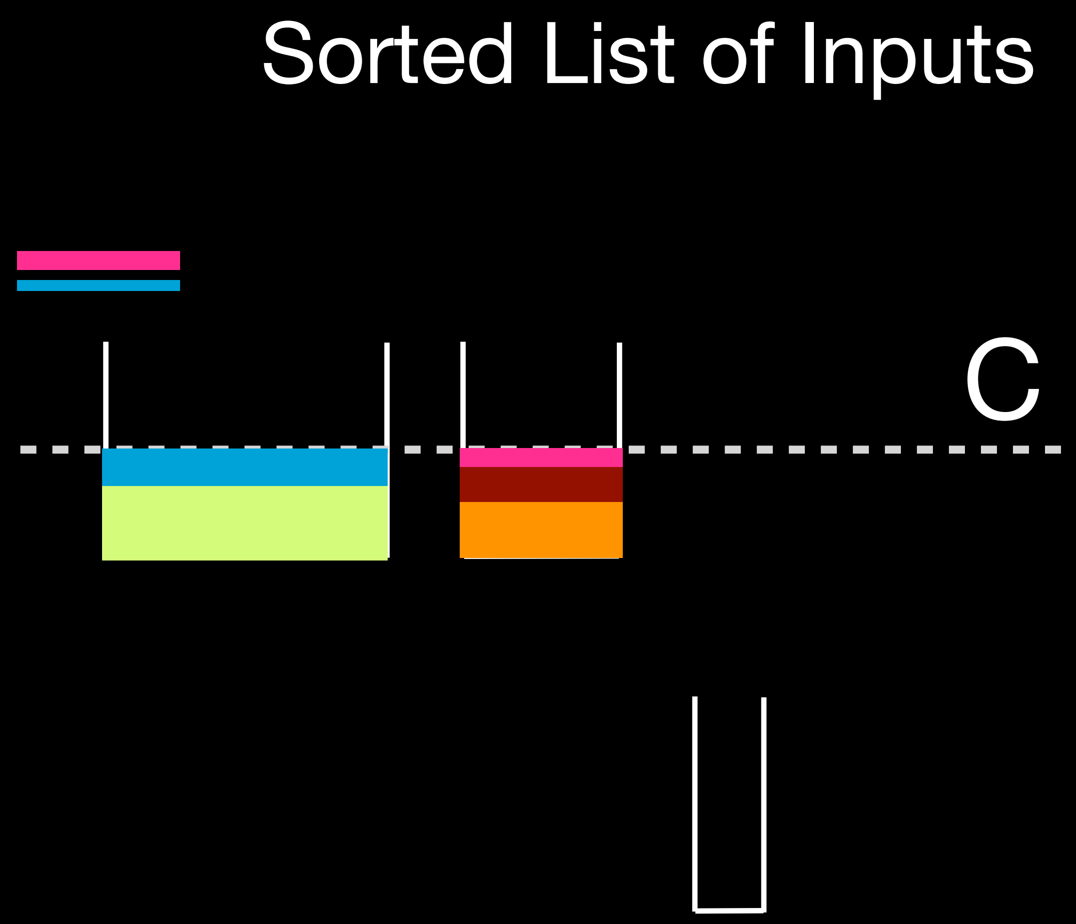
Scheduling Jobs



Scheduling Jobs

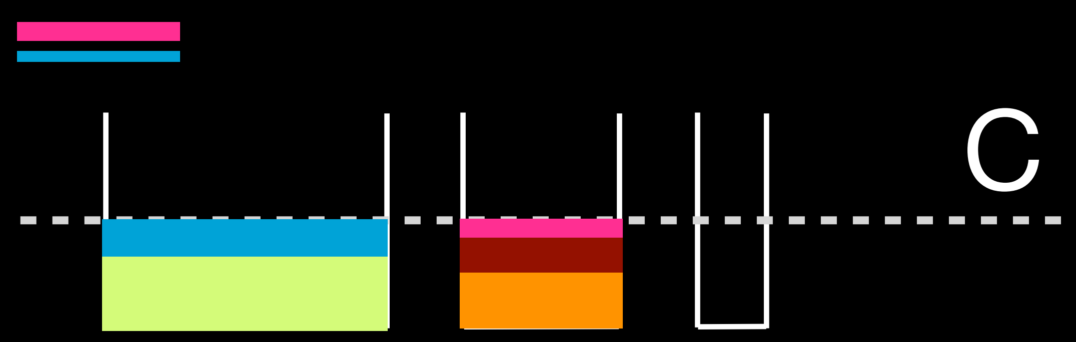


Scheduling Jobs



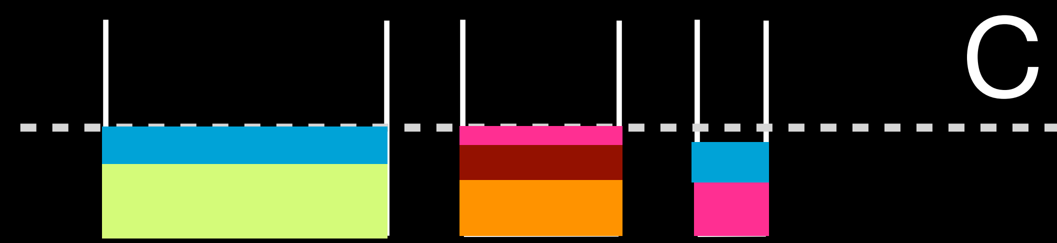
Scheduling Jobs

Sorted List of Inputs



Scheduling Jobs

Sorted List of Inputs



Scheduling Jobs

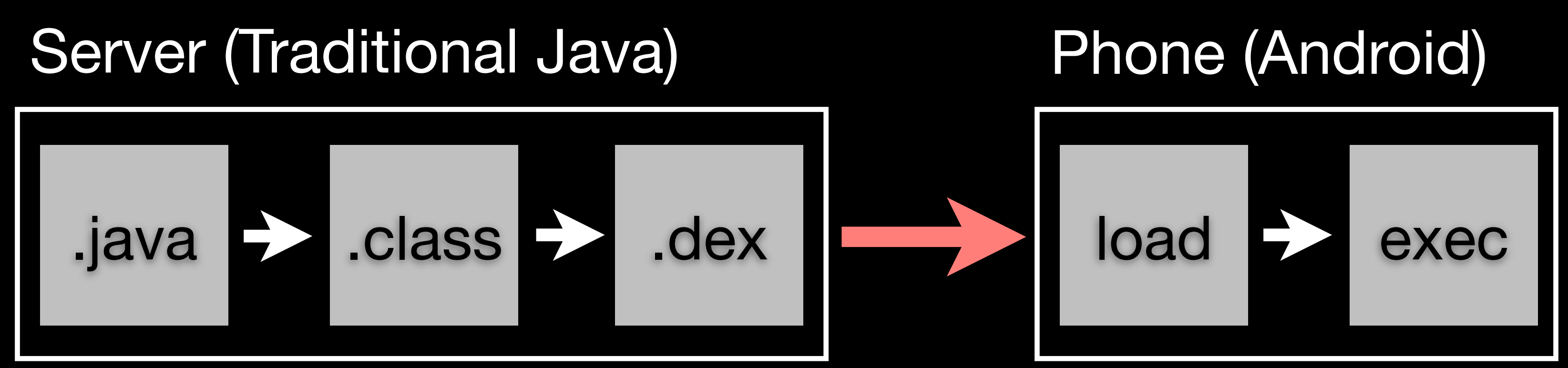
- try to produce few partitions to reduce the aggregation load at the central server, **while minimizing C**.

Automating Job Execution

- We implement the CWC service in Android
  - runs in “background” -- no human input
  - exploits the compatibility between JVM and Dalvik (a core subset of Java APIs are common)
  - leverages Java Reflection API to dynamically load classes and execute methods defined by them

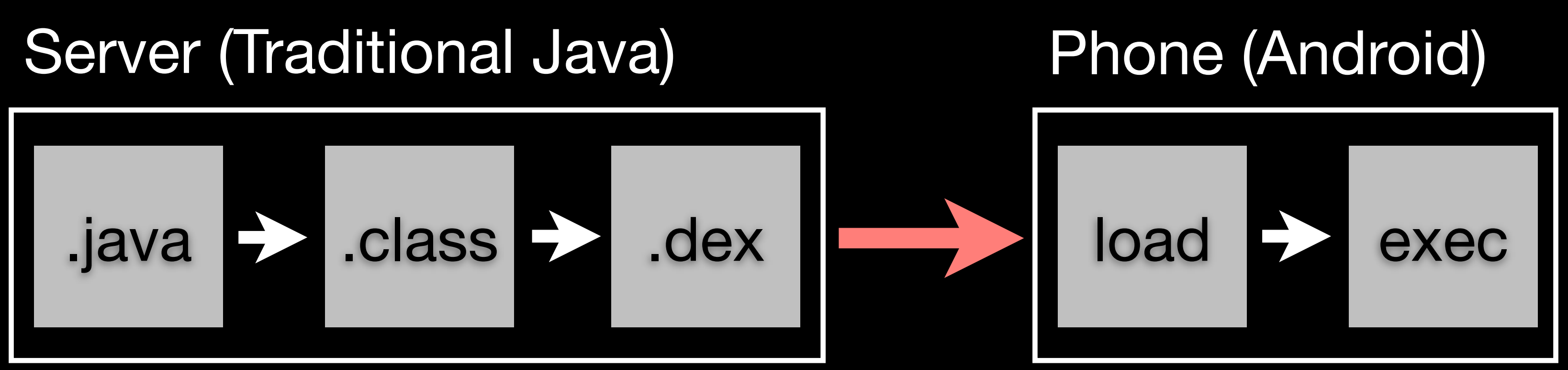
Automating Job Execution

- We implement the CWC service in Android
  - runs in “background” -- no human input
  - exploits the compatibility between JVM and Dalvik (a core subset of Java APIs are common)
  - leverages Java Reflection API to dynamically load classes and execute methods defined by them



Automating Job Execution

- We implement the CWC service in Android
  - runs in “background” -- no human input
  - exploits the compatibility between JVM and Dalvik (a core subset of Java APIs are common)
  - leverages Java Reflection API to dynamically load classes and execute methods defined by them



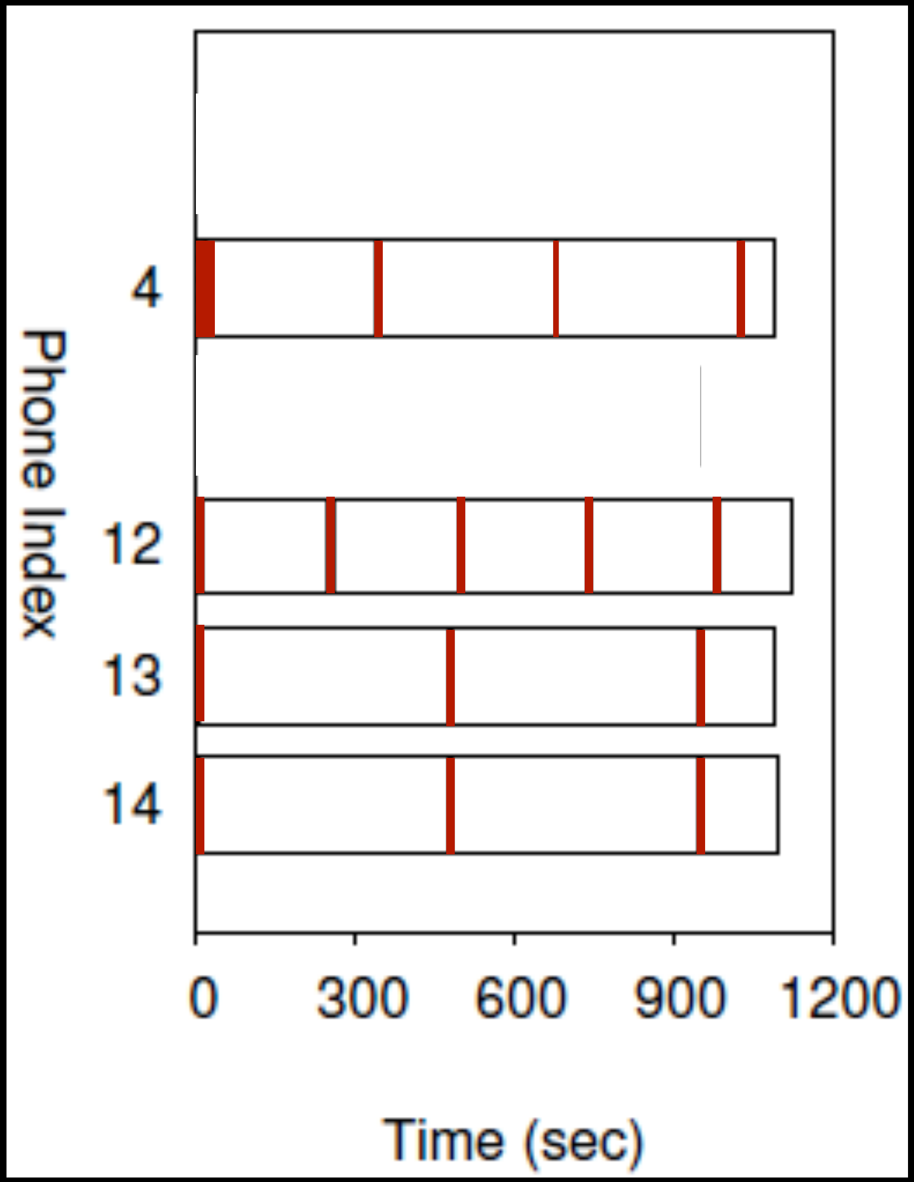
- ***The same Java code runs on both PCs and smartphones!***

Setup

Connectivity	802.11a / g, EDGE, 3G, 4G
CPU Speed	806 MHz to 1.5 GHz Single and Dual Core

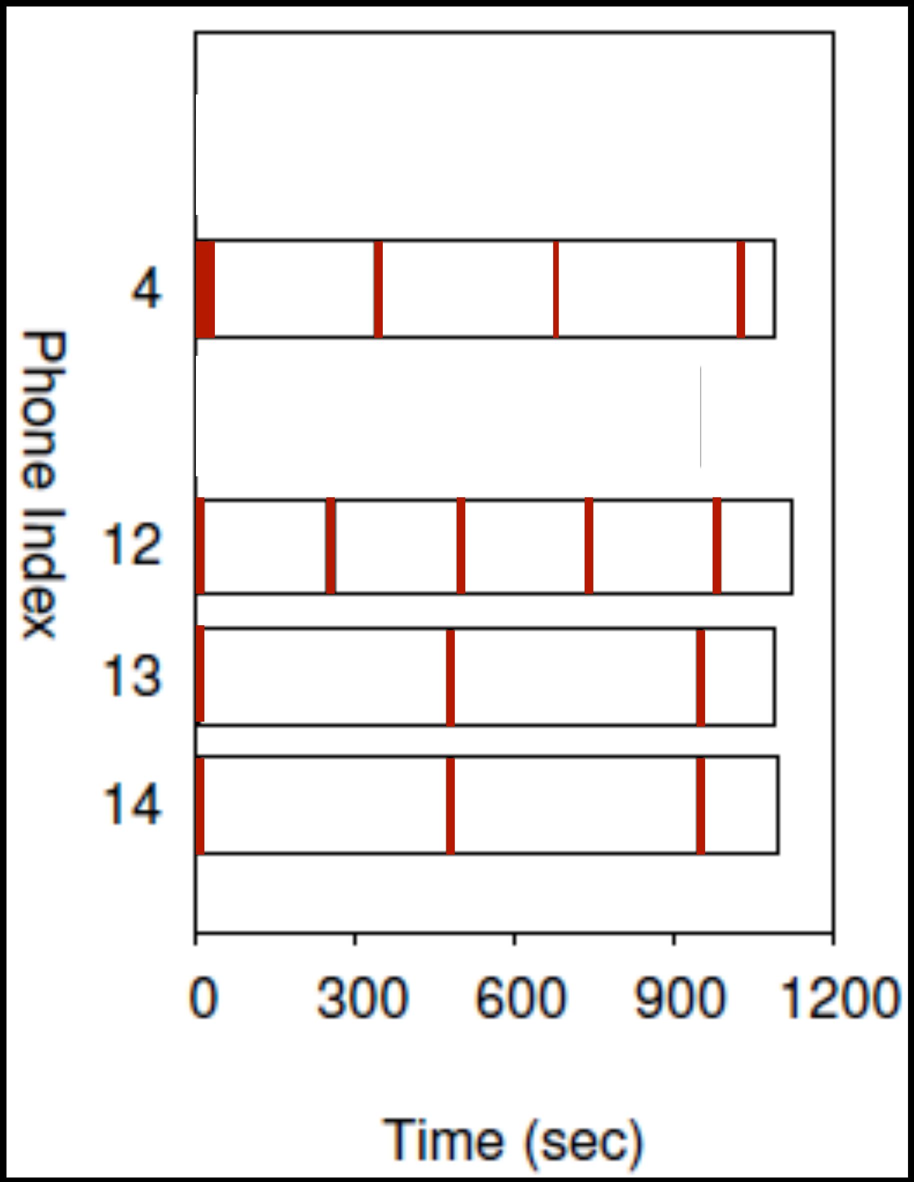
- 18 Android smartphones with CWC software.
- Lightweight central server
  - Amazon EC2 small instance (< 2 GB RAM)
  - Multi-threaded Java NIO implementation

Results



Shows a sub-set of the phones

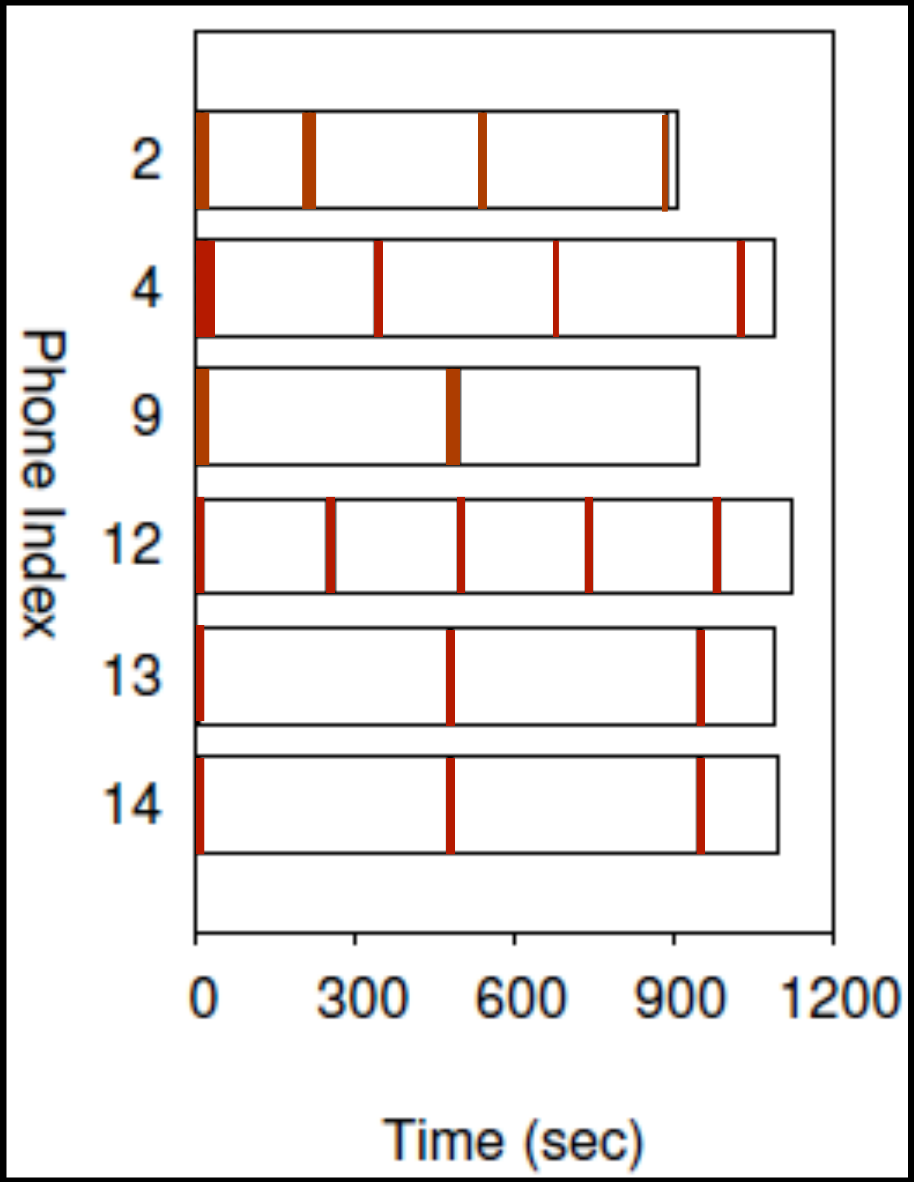
Results



Shows a sub-set of the phones

- Balanced assignment for phones 4, 12, 13, 14 (and for other phones not shown)

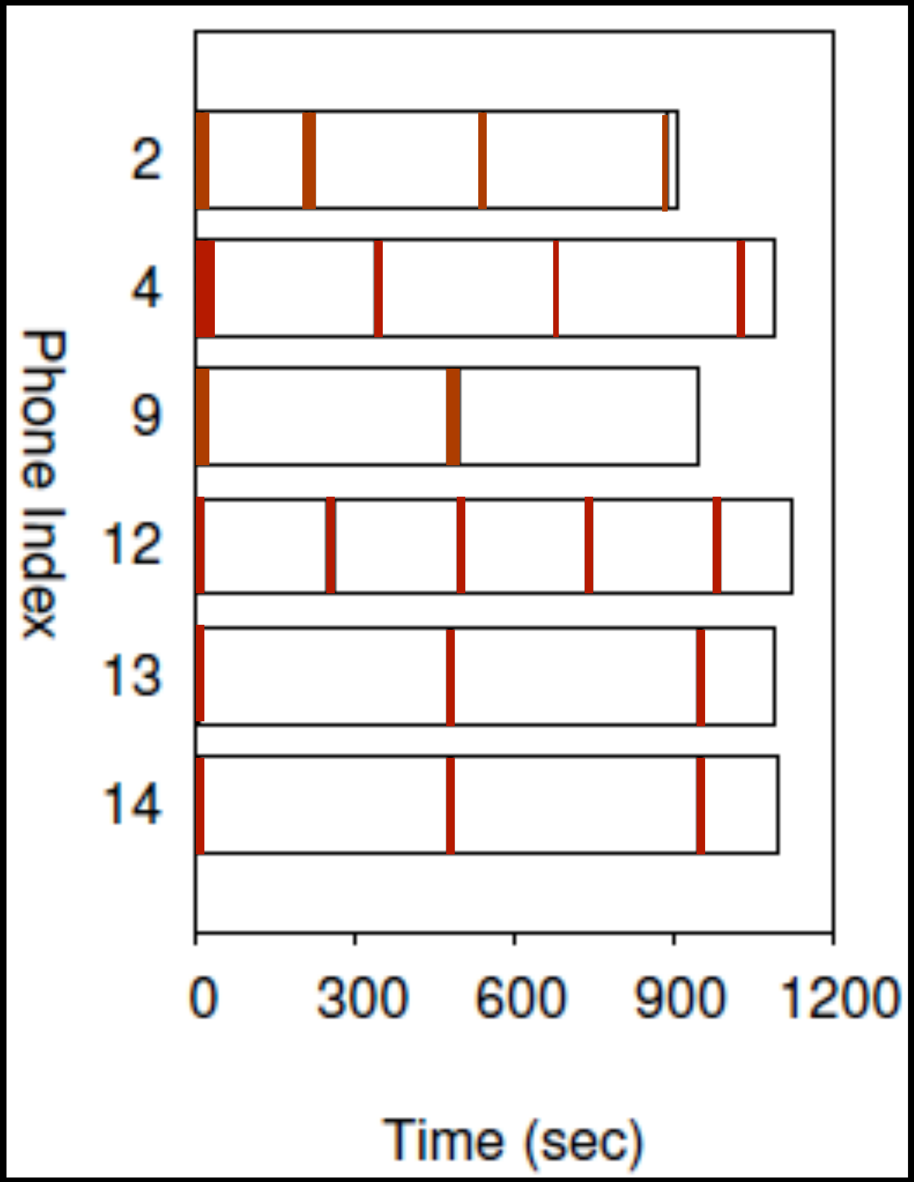
Results



Shows a sub-set of the phones

- Balanced assignment for phones 4, 12, 13, 14 (and for other phones not shown)
- Phones 2 and 9 finish earlier than others (because they are “faster” than predicted).

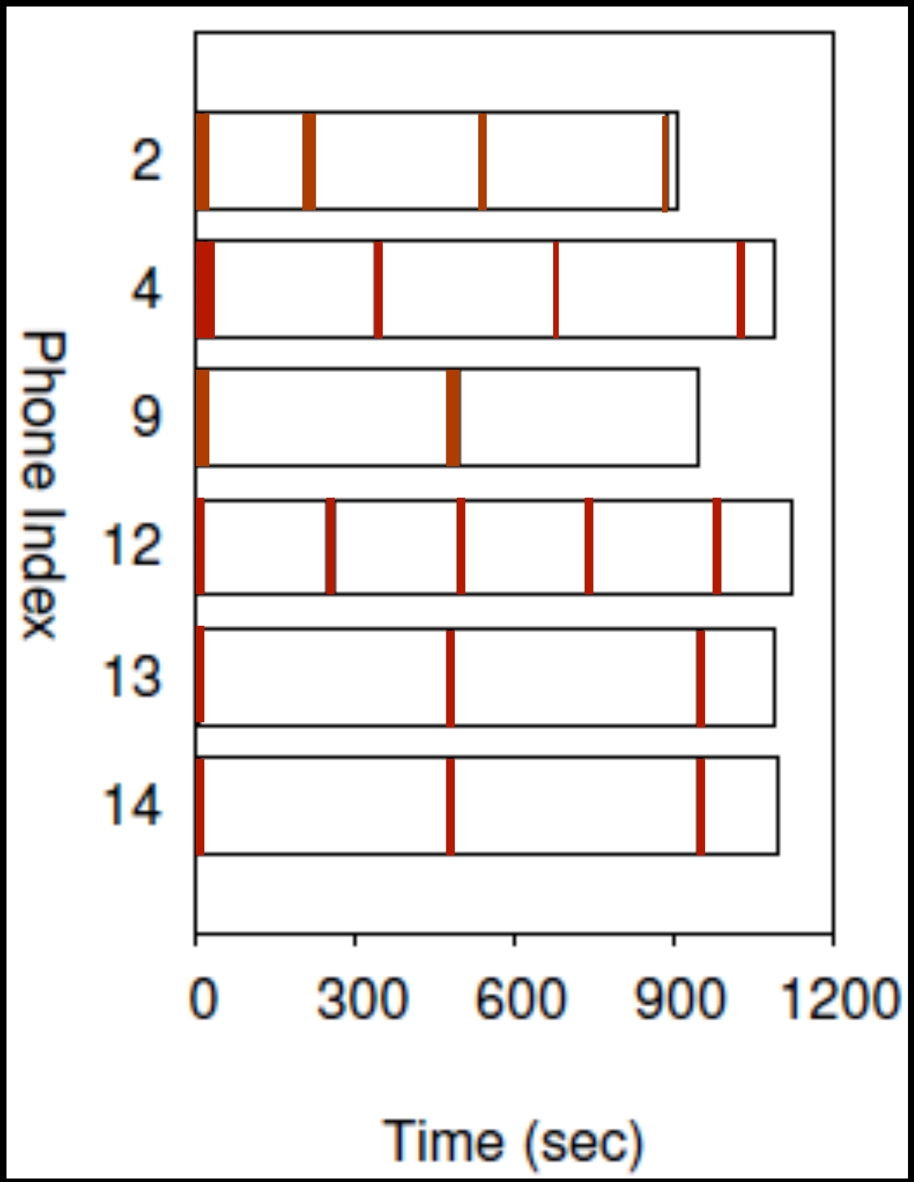
Results



Shows a sub-set of the phones

- Balanced assignment for phones 4, 12, 13, 14 (and for other phones not shown)
- Phones 2 and 9 finish earlier than others (because they are "faster" than predicted).
- Makespan is 1120 seconds.
  - %88 of the jobs are not partitioned (i.e., running on one phone), %9 have 3 partitions and %3 have 4 partitions.

Results



Shows a sub-set of the phones

- Balanced assignment for phones 4, 12, 13, 14 (and for other phones not shown)
- Phones 2 and 9 finish earlier than others (because they are “faster” than predicted).
- Makespan is 1120 seconds.
  - %88 of the jobs are not partitioned (i.e., running on one phone), %9 have 3 partitions and %3 have 4 partitions.
- How about full parallelism?
  - each job has |P| partitions (one partition per phone) -- makespan is 1720 seconds.

THANK YOU!

QUESTIONS?