Parallelism

CS 242

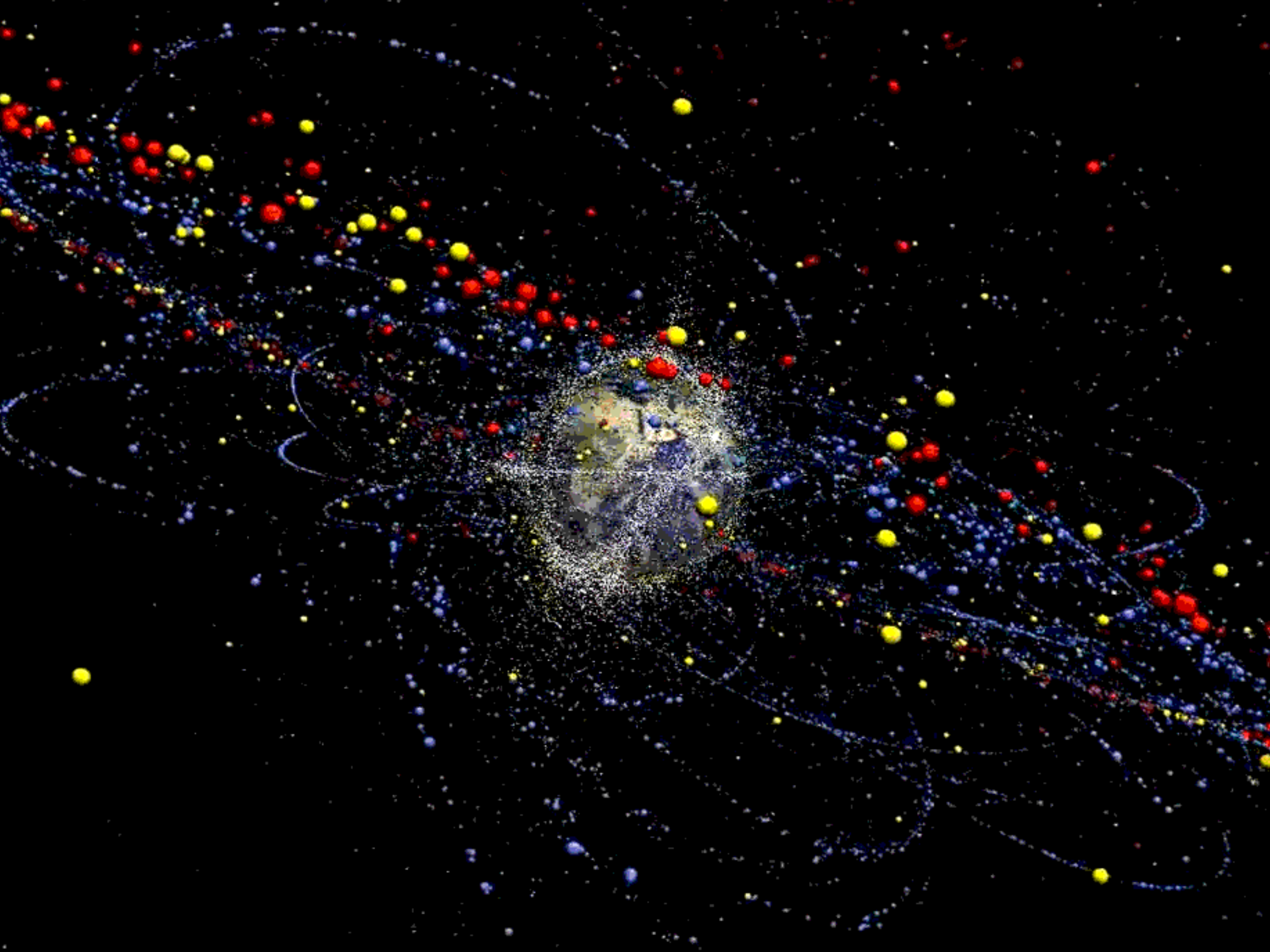
November 8, 2017

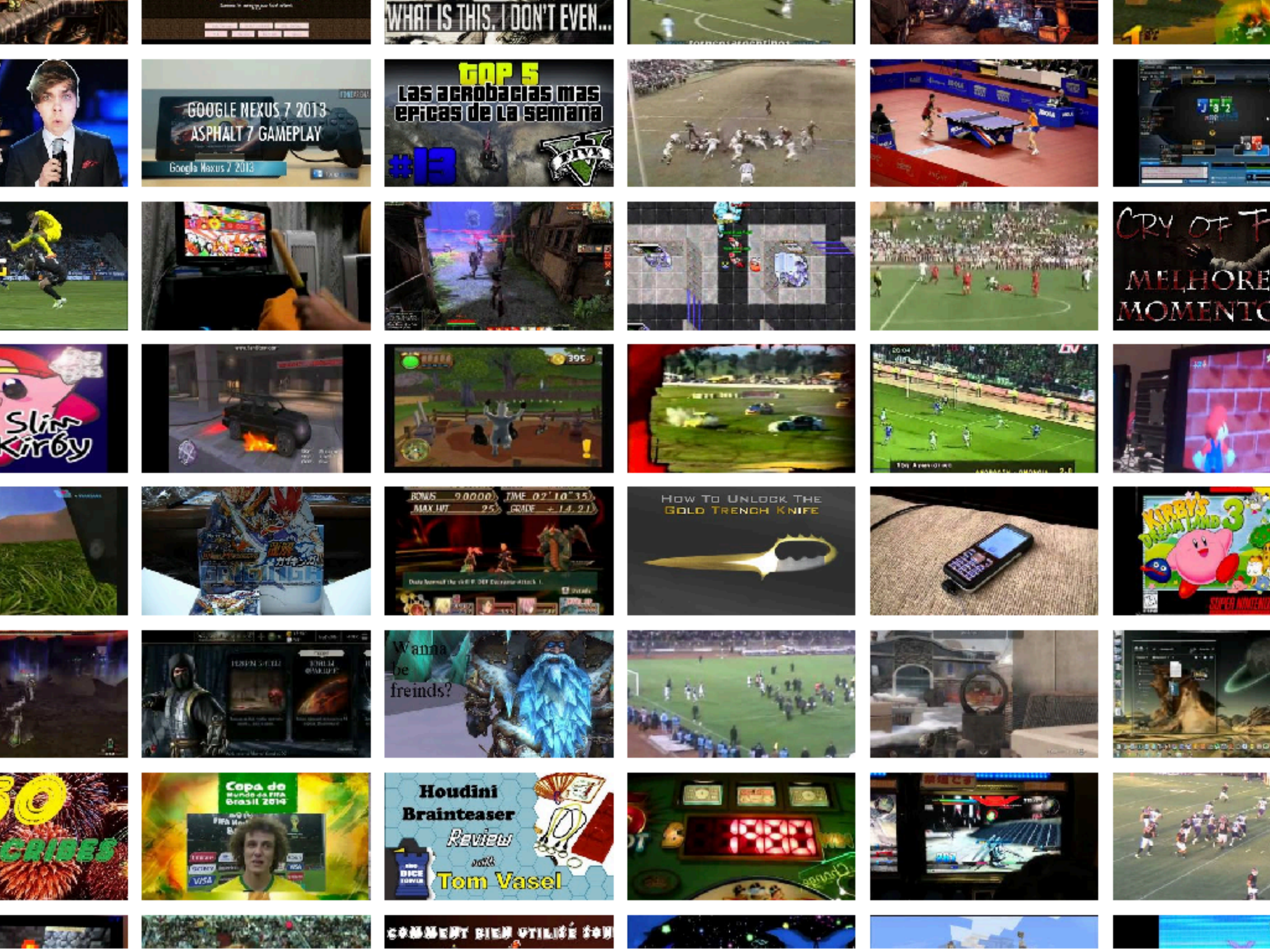
Today's goals

- Basic approach to parallel programming
- Survey of parallel programming models
- Tradeoffs in representations

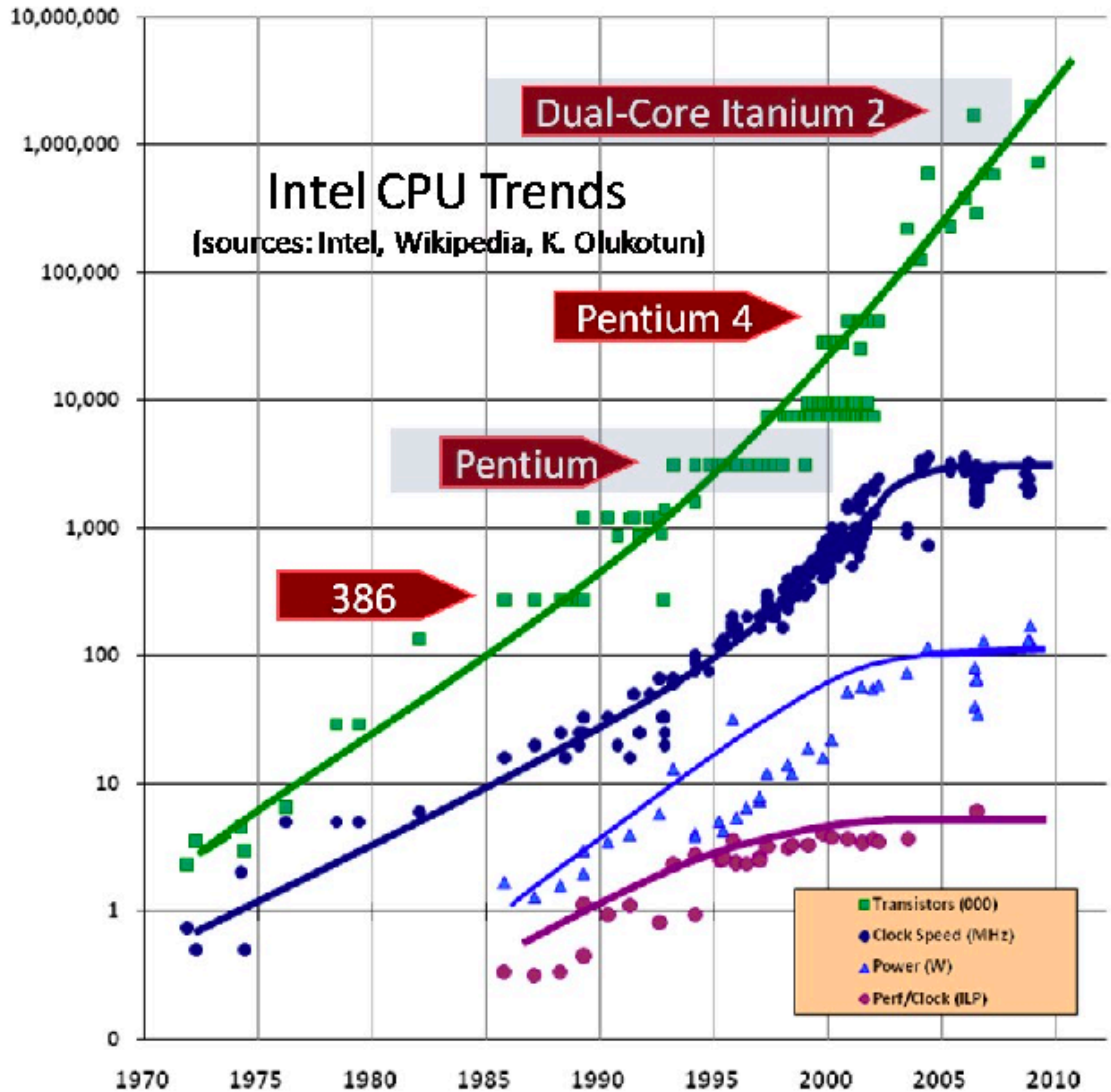
Parallelism:

Use multiple resources to accomplish a goal faster.

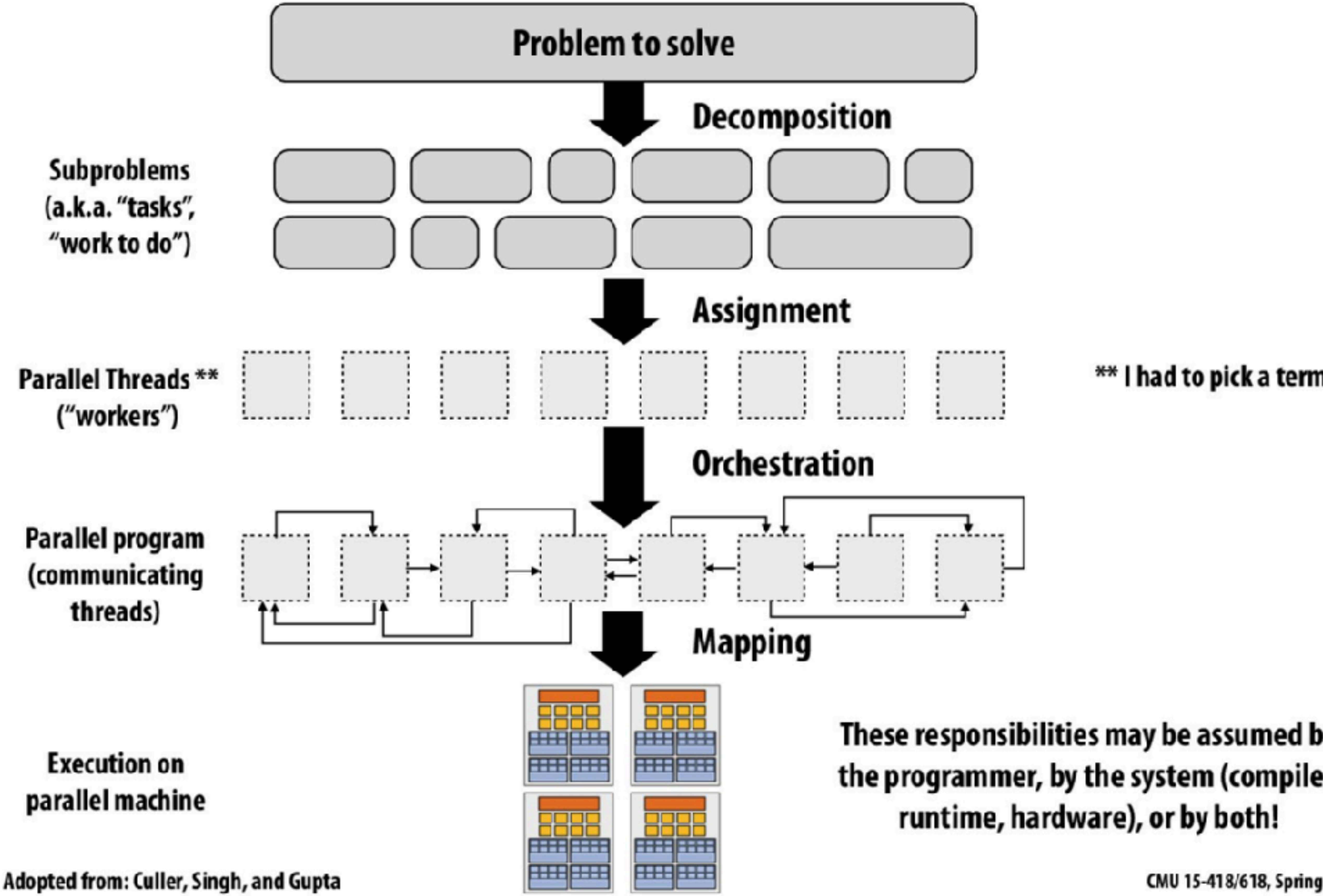



WHAT IS THIS, I DON'T EVEN...
GOOGLE NEXUS 7 2013
ASPHALT 7 GAMEPLAY
Google Nexus 7 2013
TOP 5
las acrobacias mas epicas de la semana
#13
How To Unlock The
Gold Trench Knife
Kirby's Dream Land 3
Wanna be freinds?
Copa do Mundo da FIFA Brasil 2014
Houdini
Brainteaser
Review
with
Tom Vasel

Single-core is tapped out (mostly)



Creating a parallel program



These responsibilities may be assumed by the programmer, by the system (compiler, runtime, hardware), or by both!

Adopted from: Culler, Singh, and Gupta

Sum all elements of a vector

```
fn main() {
    let vec: Vec<i64> = (0..100000).collect();

    let mut sum = 0;
    for i in vec {
        sum += i;
    }

    println!("Sum: {}", sum);
}
```

```
use std::thread;
use std::sync::Arc;

const NUM_WORKERS: usize = 8;

fn main() {
    let vec: Arc<Vec<i64>> = Arc::new((0..100000).collect());

    let chunk_size = vec.len() / NUM_WORKERS;

    let handles: Vec<thread::JoinHandle<i64>> =
        (0..NUM_WORKERS).into_iter().map(|i| {
        let vec_ref = vec.clone();
        thread::spawn(move || {
            let mut sum = 0;
            for j in (i * chunk_size)..((i + 1) * chunk_size) {
                sum += vec_ref[j];
            }
            sum
        })
    });

    let mut final_sum = 0;
    for handle in handles {
        final_sum += handle.join().unwrap();
    }

    println!("Sum: {}", final_sum);
}
```

1. Decomposition: reduction

2. Assignment

3. Orchestration

4. Mapping

OpenMP parallelizes for loops on CPU

```c
int main() {
  int x[] = {1, 2, 3, 4, 5};

  #pragma omp parallel for
  for (int i = 0; i < 5; ++i) {
    x[i] = x[i] + 1;
  }
}
```

"Each iteration of this loop is independent"

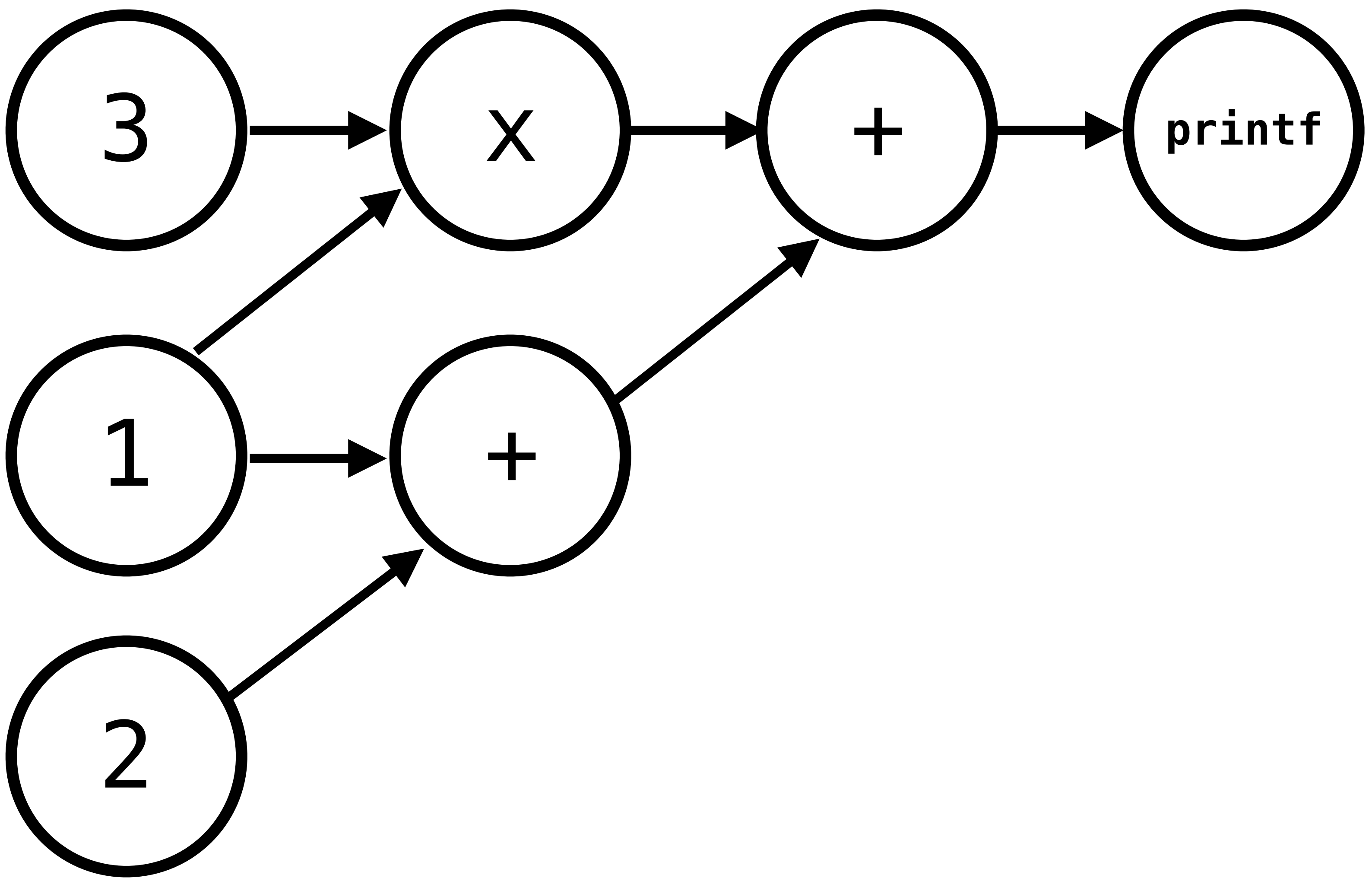
CUDA parallelizes functions on the GPU

```
__global__ void add_one(int *x) {
  int index = blockIdx.x * blockDim.x + threadIdx.x;
  x[index] = x[index] + 1;
}

int main() {
  int x[256];
  int* x_gpu;
  cudaMalloc(&x_gpu, 256 * sizeof(int));
  cudaMemcpy(x_gpu, x, 256 * sizeof(int),
             cudaMemcpyHostToDevice);

  add_one<<<1, 256>>>(x_gpu);
}
```

"Each call this function is independent"

Problem: most of the hard work is left to the programmer.

```c
int x = 1;
int y = x + 2;
int z = x * 3;
printf("%d\n", z + y)
```



TensorFlow = dataflow + tensors

```python
import tensorflow as tf

a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)

c = a + b
d = a * c

with tf.Session() as sess:
    result = sess.run([d], {a: 2, b: 3})
    print(result)
```

TensorFlow = dataflow + tensors

```python
import tensorflow as tf

x = tf.constant([[1.0, 2.0],
                 [3.0, 4.0]])
y = tf.constant([[1.0, 0.0],
                 [0.0, 1.0]])
z = tf.matmul(x, y)

with tf.Session() as sess:
    print(sess.run(z))
```

Spark = RDDFlow (RDD[T] = Vec<T>)

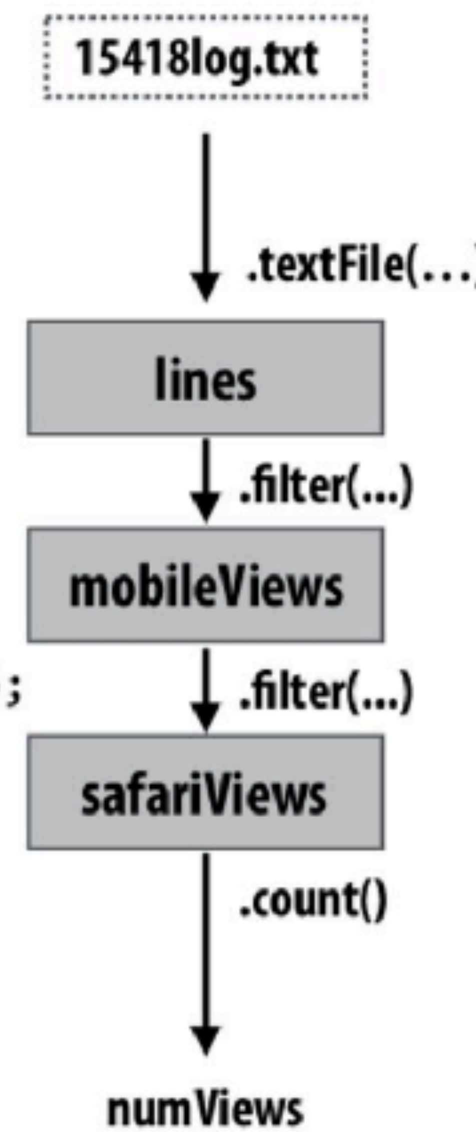
Spark's key programming abstraction:

- Read-only collection of records (immutable)
- RDDs can only be created by deterministic ***transformations*** on data in persistent storage or on existing RDDs
- ***Actions*** on RDDs return data to application

RDDs

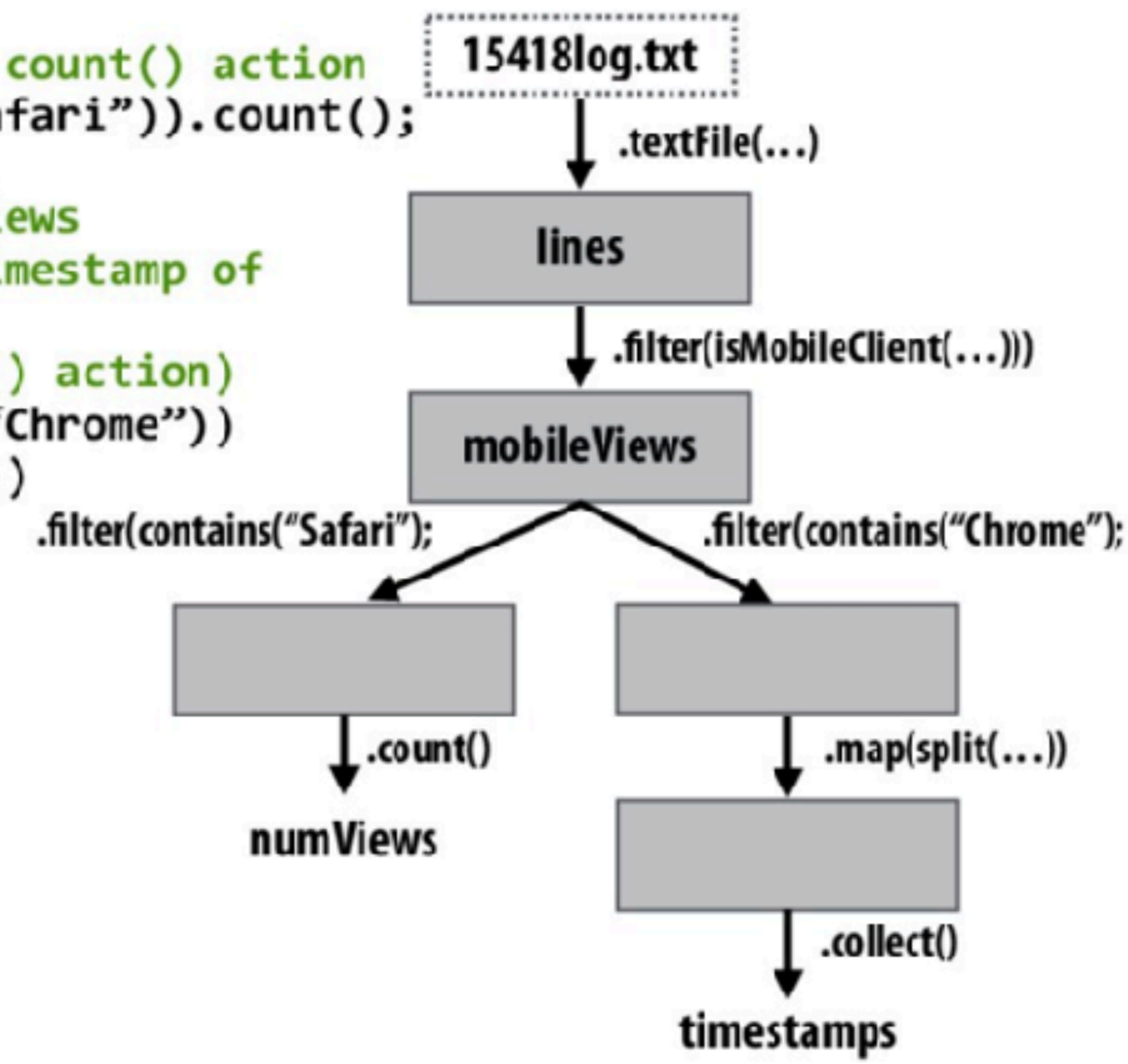
```
// create RDD from file system data
var lines = spark.textFile("hdfs://15418log.txt");

// create RDD using filter() transformation on lines
var mobileViews = lines.filter((x: String) => isMobileClient(x));

// another filter() transformation
var safariViews = mobileViews.filter((x: String) => x.contains("Safari"));

// then count number of elements in RDD via count() action
var numViews = safariViews.count();
```

int



Spark supports DAGs

```
// create RDD from file system data
var lines = spark.textFile("hdfs://15418log.txt");

// create RDD using filter() transformation on lines
var mobileViews = lines.filter((x: String) => isMobileClient(x));

// instruct Spark runtime to try to keep mobileViews in memory
mobileViews.persist();

// create a new RDD by filtering mobileViews
// then count number of elements in new RDD via count() action
var numViews = mobileViews.filter(_.contains("Safari")).count();

// 1. create new RDD by filtering only Chrome views
// 2. for each element, split string and take timestamp of
//    page view
// 3. convert RDD to a scalar sequence (collect() action)
var timestamps = mobileViews.filter(_.contains("Chrome"))
                            .map(_.split(" ")(0))
                            .collect();
```



Spark transformations and actions

Transformations: (data parallel operators taking an input RDD to a new RDD)

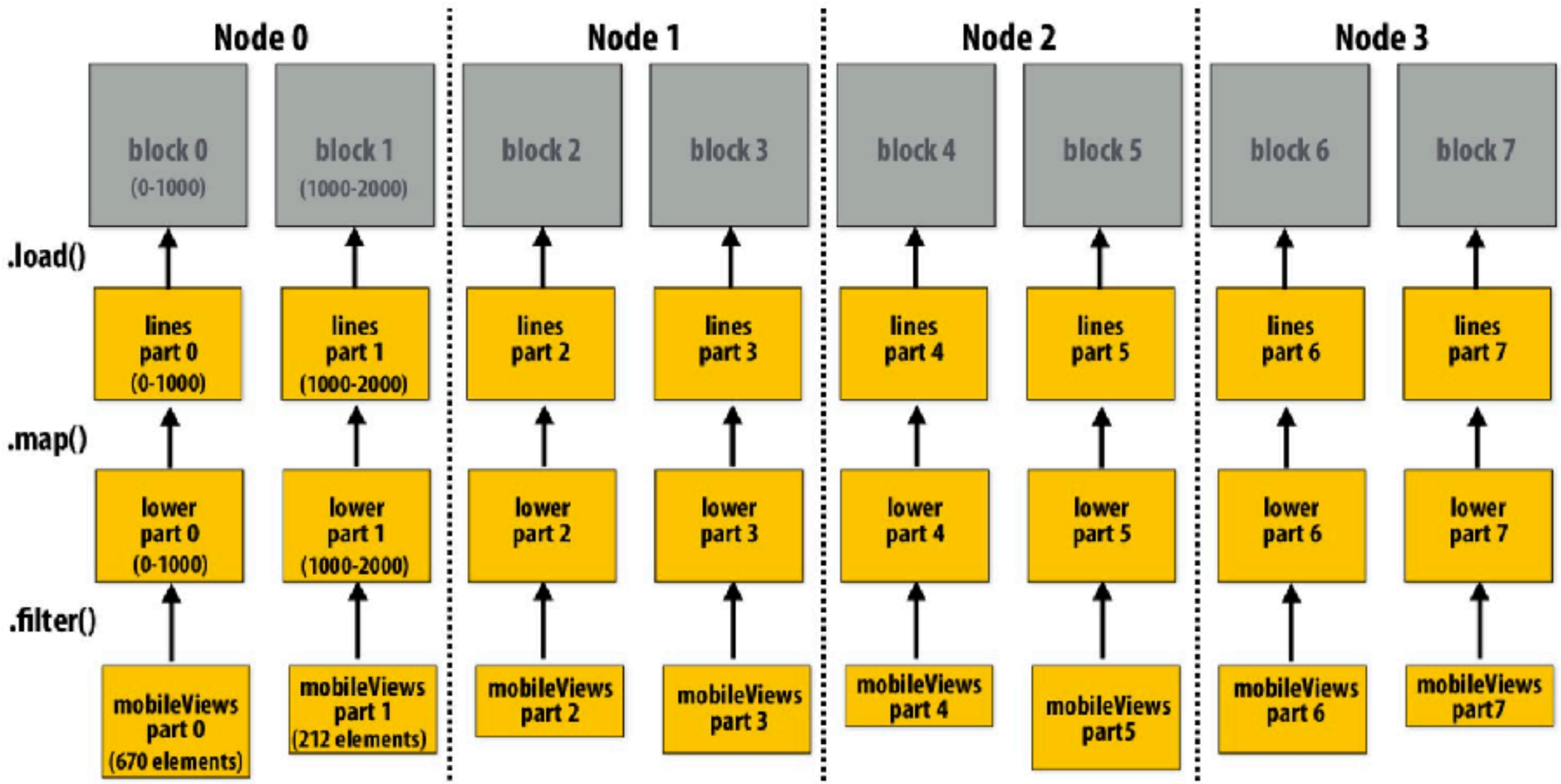
$map(f : \mathrm{T} \Rightarrow \mathrm{U})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]}$
$filter(f : \mathrm{T} \Rightarrow \mathrm{Bool})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]}$
$flatMap(f : \mathrm{T} \Rightarrow \mathrm{Seq[U]})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]}$
$sample(fraction : \mathrm{Float})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]}$ (Deterministic sampling)
$groupByKey()$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, Seq[V])]}$
$reduceByKey(f : (\mathrm{V}, \mathrm{V}) \Rightarrow \mathrm{V})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$
$union()$	:	$(\mathrm{RDD[T]}, \mathrm{RDD[T]}) \Rightarrow \mathrm{RDD[T]}$
$join()$	:	$(\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (V, W))]}$
$cogroup()$	:	$(\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (Seq[V], Seq[W]))]}$
$crossProduct()$	:	$(\mathrm{RDD[T]}, \mathrm{RDD[U]}) \Rightarrow \mathrm{RDD[(T, U)]}$
$mapValues(f : \mathrm{V} \Rightarrow \mathrm{W})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, W)]}$ (Preserves partitioning)
$sort(c : \mathrm{Comparator[K]})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$
$partitionBy(p : \mathrm{Partitioner[K]})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$

Actions: (provide data back to the "host" application)

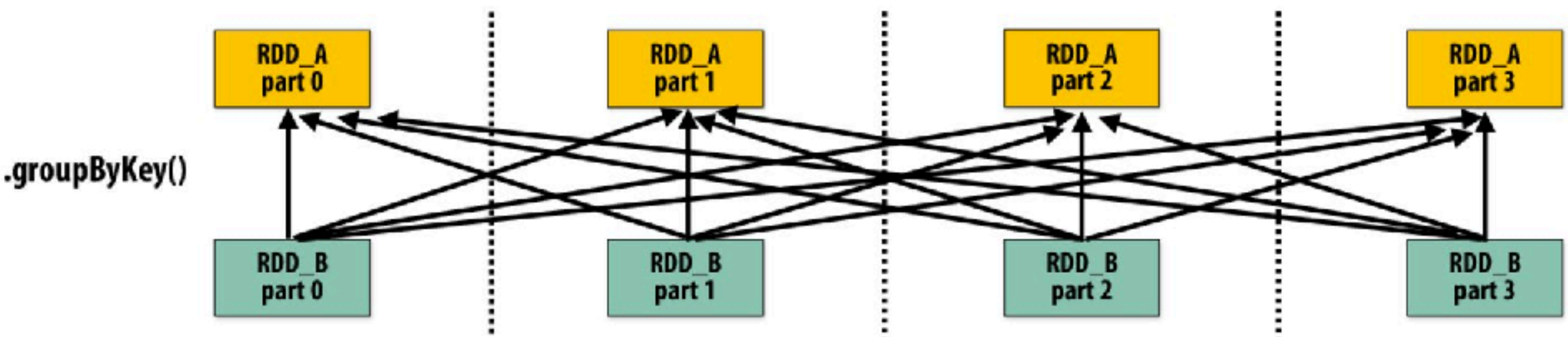
$count()$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{Long}$
$collect()$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{Seq[T]}$
$reduce(f : (\mathrm{T}, \mathrm{T}) \Rightarrow \mathrm{T})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{T}$
$lookup(k : \mathrm{K})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{Seq[V]}$ (On hash/range partitioned RDDs)
$save(path : \mathrm{String})$	:	Outputs RDD to a storage system, *e.g.*, HDFS

Solution #1: partitioning

```
var lines = spark.textFile("hdfs://15418log.txt");
var lower = lines.map(_.toLower());
var mobileViews = lower.filter(x => isMobileClient(x));
var howMany = mobileViews.count();
```



Solution #1: partitioning



Solution #2: streaming

```
fn main() {
    // Non-streaming
    let v = (0..(1024i64*1024*1024*1024)).into_iter();
    let v1: Vec<i64> = v.collect();
    let mut v2 = Vec::new();
    for x in v1 {
        v2.push(x + 1);
    }
    println!("{}", v2[0]);

    // Streaming
    let v = (0..(1024i64*1024*1024*1024)).into_iter();
    let mut v2 = v.map(|x| x + 1);
    println!("{}", v2.next().unwrap());
}
```