Parallel Programming
Exercise Session 5

Spring 2020

Feedback: Exercise 4

Pipelining

Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

Task a) total time if strictly sequential order?

Pipelining

Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

Task a) total time if strictly sequential order?

50 + 90 + 15

155

Pipelining

Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

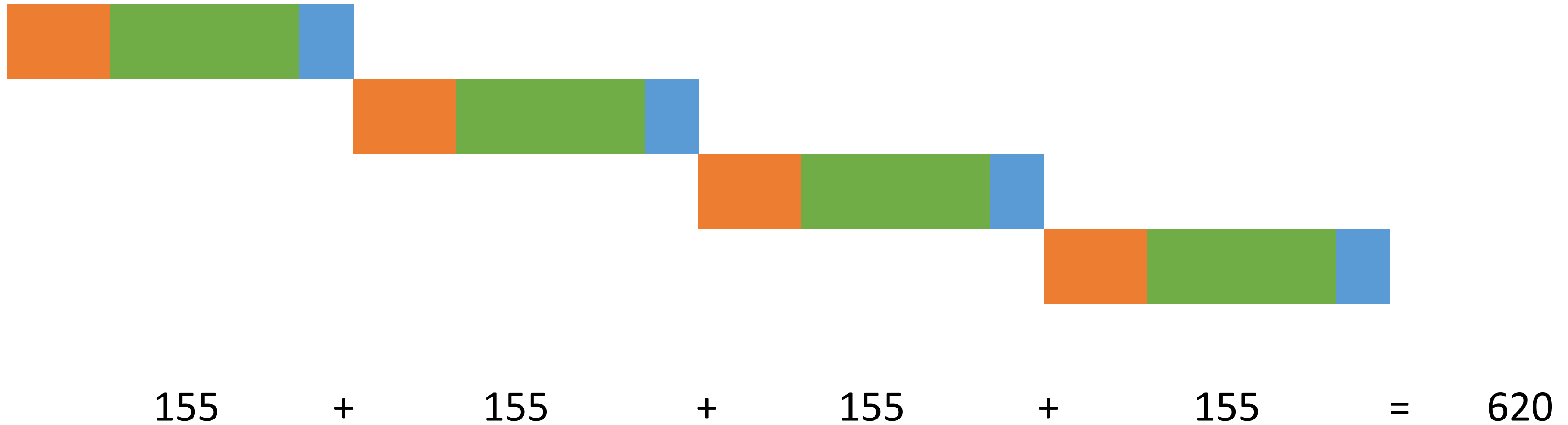
Task a) total time if strictly sequential order?

155 + 155

Pipelining

Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

Task a) total time if strictly sequential order?



Pipelining

Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

Task b) what would be a better (faster) strategy?

Pipelining

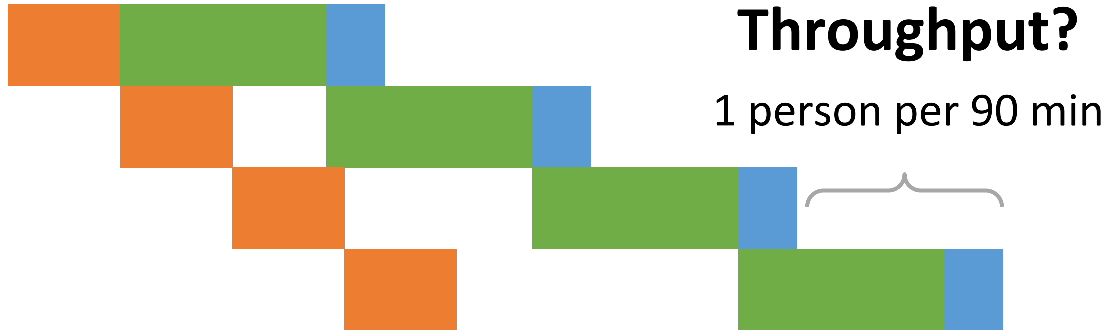
Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

Task b) what would be a better (faster) strategy?

Pipelining

Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

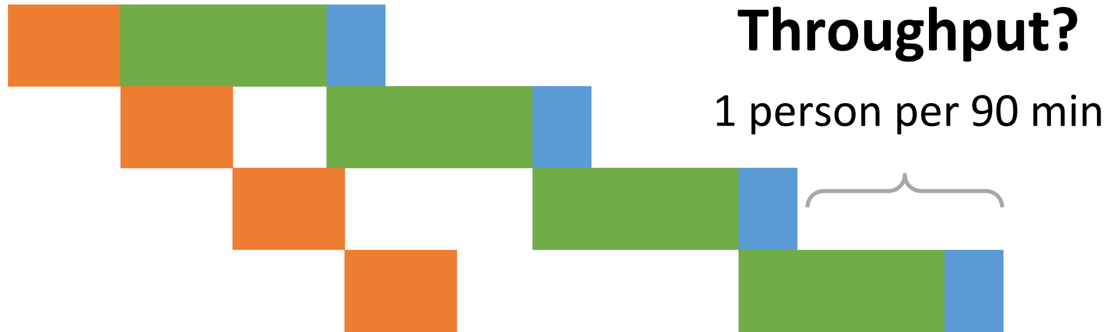
Task b) what would be a better (faster) strategy?



Pipelining

Washing - 50 min, Dryer - 90 min, Iron - 15 min

Task b) what would be a better (faster) strategy?



How to compute throughput fast for pipelines with no duplicated stages?

$$\frac{1}{\max(computationtime(stages))}$$

Pipelining

Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

Task c) what if they bought another **dryer**?

Pipelining

Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

Task c) what if they bought another **dryer**?

Pipelining

Washing - 50 min, Dryer - 90 min, Iron - 15 min

Task c) what if they bought another **dryer**?

Latency?

155 min

Throughput?

1 person per 50 min

Pipelining

Washing - 50 min, **Dryer** - 90 min, **Iron** - 15 min

Task c) what if they bought another **dryer**?

Pipeline is balanced even though the stages do not take the same time.

all stages require the same time $\Rightarrow$ balanced

but

balanced $\nRightarrow$ all stages require the same time

Pipelining II

```
for (int i = 0; i < size; i++) {
  data[i] = data[i] * data[i];
}
```

```
for (int i = 0; i < size; i += 2) {
  j = i + 1;
  data[i] = data[i] * data[i];
  data[j] = data[j] * data[j];
}
```

```
for (int i = 0; i < size; i += 4) {
  j = i + 1;
  k = i + 2;
  l = i + 3;
  data[i] = data[i] * data[i];
  data[j] = data[j] * data[j];
  data[k] = data[k] * data[k];
  data[l] = data[l] * data[l];
}
```

Pipelining II

```
for (int i = 0; i < size; i++) {
  data[i] = data[i] * data[i];
}
```

```
for (int i = 0; i < size; i += 2) {
  j = i + 1;
  data[i] = data[i] * data[i];
  data[j] = data[j] * data[j];
}
```

```
for (int i = 0; i < size; i += 4) {
  j = i + 1;
  k = i + 2;
  l = i + 3;
  data[i] = data[i] * data[i];
  data[j] = data[j] * data[j];
  data[k] = data[k] * data[k];
  data[l] = data[l] * data[l];
}
```

Assumptions:

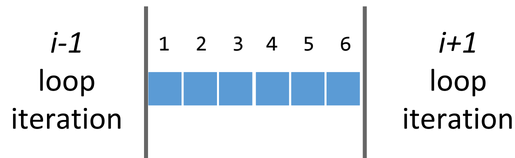
- Only one instruction can be issued per cycle.
- Loop body computation must be fully finished before next iteration starts
- Consider only arithmetic expressions in loop body and ignore all other operations (i.e., loads, stores, loop counter increment, etc.)
- Consider only *execute* step of an instruction (e.g., ignore fetch, decode, etc.)

How many cycles does the processor need to execute the following loops?

- Addition takes 3 cycles to execute
- Multiplication takes 6 cycles to execute

Pipelining II

```
for (int i = 0; i < size; i++) {
  data[i] = data[i] * data[i];
}
```

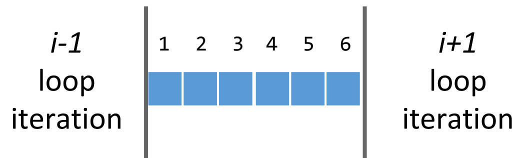



```
for (int i = 0; i < size; i += 2) {
  j = i + 1;
  data[i] = data[i] * data[i];
  data[j] = data[j] * data[j];
}
```

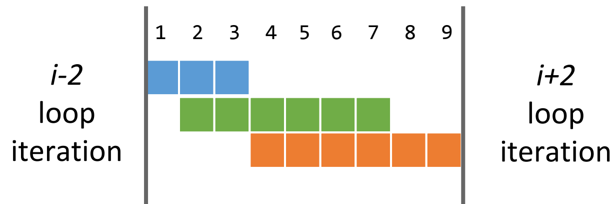
```
for (int i = 0; i < size; i += 4) {
  j = i + 1;
  k = i + 2;
  l = i + 3;
  data[i] = data[i] * data[i];
  data[j] = data[j] * data[j];
  data[k] = data[k] * data[k];
  data[l] = data[l] * data[l];
}
```

Pipelining II

```
for (int i = 0; i < size; i++) {
  data[i] = data[i] * data[i];
}
```




```
for (int i = 0; i < size; i += 2) {
  j = i + 1;
  data[i] = data[i] * data[i];
  data[j] = data[j] * data[j];
}
```

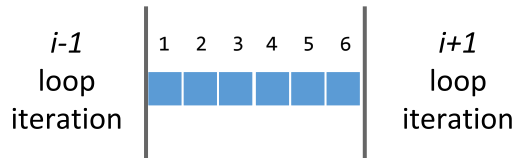



```
for (int i = 0; i < size; i += 4) {
  j = i + 1;
  k = i + 2;
  l = i + 3;
  data[i] = data[i] * data[i];
  data[j] = data[j] * data[j];
  data[k] = data[k] * data[k];
  data[l] = data[l] * data[l];
}
```

Pipelining II

```
for (int i = 0; i < size; i++) {
  data[i] = data[i] * data[i];
}
```

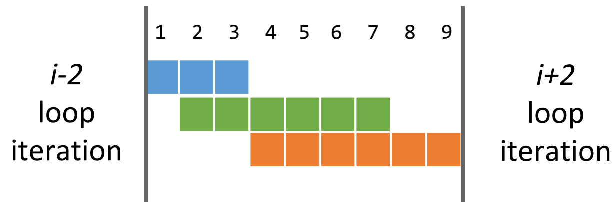
i-1 loop iteration



i+1 loop iteration

```
for (int i = 0; i < size; i += 2) {
  j = i + 1;
  data[i] = data[i] * data[i];
  data[j] = data[j] * data[j];
}
```

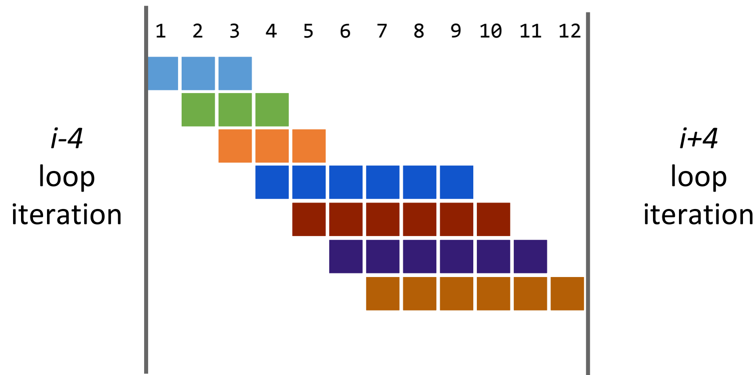
i-2 loop iteration



i+2 loop iteration

```
for (int i = 0; i < size; i += 4) {
  j = i + 1;
  k = i + 2;
  l = i + 3;
  data[i] = data[i] * data[i];
  data[j] = data[j] * data[j];
  data[k] = data[k] * data[k];
  data[l] = data[l] * data[l];
}
```

i-4 loop iteration



i+4 loop iteration

Loop Parallelism

Can we parallelize the following loop?

```
for (int i=1; i<size; i++) { // for loop: i from 1 to (size-1)
  if (data[i-1] > 0)          // If the previous value is positive
    data[i] = (-1)*data[i];   // change the sign of this value
}                             // end for loop
```

Loop Parallelism

Can we parallelize the following loop?

```
for (int i=1; i<size; i++) { // for loop: i from 1 to (size-1)
  if (data[i-1] > 0)          // If the previous value is positive
    data[i] = (-1)*data[i];   // change the sign of this value
}                             // end for loop
```

```
for (int i = 0; i < size; i++) {    // for loop: i from 0 to (size-1)
  data[i] = Math.sin(data[i]);      // calculate sin() of the value
}                                   // end for loop
```

Supplementary Exercise 4

SOLA Stafette

- 14 runners (numbered from 0 to 13)
- each runner can start after the previous runner finished (except the first one).

Under which assumption the following code works?

```
public class RunnerThread extends Thread {

    private AtomicInteger y;
    private int id;

    public RunnerThread(int id, AtomicInteger y) {
      this.id = id;
      this.y = y;
    }

    public void run(){
      while (y.get() != this.id) {
        // warten bis ich an der Reihe bin / wait until it is my turn
      }

      // laufen / do running
      y.incrementAndGet();
    }
 }
```

Initial value of y?

0

Reference to object y?

needs to be the same object shared across all the threads

SOLA Stafette

- 14 runners (numbered from 0 to 13)
- each runner can start after the previous runner finished (except the first one).

Under which assumption the following code works?

Initial value of y?

0

Reference to object y?

needs to be the same object shared across all the threads

```
public class RunnerThread extends Thread {

    private AtomicInteger y;
    private int id;

    public RunnerThread(int id, AtomicInteger y) {
      this.id = id;
      this.y = y;
    }

    public void run(){
      while (y.get() != this.id) {
        // warten bis ich an der Reihe bin / wait until it is my turn
      }

      // laufen / do running
      y.incrementAndGet();
    }
}
```

Ensures that `y.get()` returns the updated value after calling `y.incrementAndGet();`

SOLA Stafette

Complete the implementation using wait/notify

```
public class WaitNotifyRunnerThread extends Thread {

    private .................. x;
    private int id;

    public WaitNotifyRunnerThread(int id, .................. x) {
        this.id = id;
        this.x = x;
    }

    public ................. void run(){
        ...................................................
        ...................................................
        ...................................................
        ...................................................
        ...................................................
        ...................................................
        ...................................................
        ...................................................
    }
}
```

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?

```
public class WaitNotifyRunnerThread extends Thread {

    private .................. x;
    private int id;

    public WaitNotifyRunnerThread(int id, .................. x) {
        this.id = id;
        this.x = x;
    }

    public ................. void run(){
        synchronized (?) {
            ...................................................
            ...................................................
            ...................................................
            ...................................................
            ...................................................
            ...................................................
            ...................................................
    }
}
```

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?
3. Check that it's our turn, wait otherwise

```
public class WaitNotifyRunnerThread extends Thread {

    private .................. x;
    private int id;

    public WaitNotifyRunnerThread(int id, .................. x) {
        this.id = id;
        this.x = x;
    }

    public ................. void run(){
        synchronized (?) {
            while (condition) {
                ?.wait();
            }
            ...................................................
            ...................................................
            ...................................................
            ...................................................
    }
}
```

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?
3. Check that it's our turn, wait otherwise
4. Do work (run)

```
public class WaitNotifyRunnerThread extends Thread {

    private .................. x;
    private int id;

    public WaitNotifyRunnerThread(int id, .................. x) {
        this.id = id;
        this.x = x;
    }

    public ................. void run(){
        synchronized (?) {
            while (condition) {
                ?.wait();
            }
            // laufen / do running
        ...................................................
        ...................................................
        ...................................................
    }
}
```

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?
3. Check that it's our turn, wait otherwise
4. Do work (run)
5. Update the condition

```
public class WaitNotifyRunnerThread extends Thread {

    private .................. x;
    private int id;

    public WaitNotifyRunnerThread(int id, .................. x) {
        this.id = id;
        this.x = x;
    }

    public .................. void run(){
        synchronized (?) {
            while (condition) {
                ?.wait();
            }
            // laufen / do running
            // update the condition
        ..................................................
        ..................................................
    }
}
```

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?
3. Check that it's our turn, wait otherwise
4. Do work (run)
5. Update the condition
6. Notify waiting threads

This is a basic pattern that you will see in many tasks. Now, let's fill in the details.

```
public class WaitNotifyRunnerThread extends Thread {

    private .................. x;
    private int id;

    public WaitNotifyRunnerThread(int id, .................. x) {
        this.id = id;
        this.x = x;
    }

    public ................. void run(){
        synchronized (?) {
            while (condition) {
                ?.wait();
            }
            // laufen / do running
            // update the condition
            ?.notify(); // or notifyAll()
        }
    }
}
```

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. **What to synchronize on?**
3. Check that it's our turn, wait otherwise
4. Do work (run)
5. Update the condition
6. Notify waiting threads

```
public class WaitNotifyRunnerThread extends Thread {

    private .................. x;
    private int id;

    public WaitNotifyRunnerThread(int id, .................. x) {
        this.id = id;
        this.x = x;
    }

    public ................. void run(){
        synchronized (?) {
            while (condition) {
                 ?.wait();
            }
            // laufen / do running
            // update the condition
           ?.notify(); // or notifyAll()
        }
    }
}
```

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. **What to synchronize on?**
3. Check that it's our turn, wait otherwise
4. Do work (run)
5. Update the condition
6. Notify waiting threads

```
public class WaitNotifyRunnerThread extends Thread {

    private .................. x;
    private int id;

    public WaitNotifyRunnerThread(int id, .................. x) {
        this.id = id;
        this.x = x;
    }

    public ................. void run(){
        synchronized (x) {
            while (condition) {
                x.wait();
            }
            // laufen / do running
           // update the condition
          x.notify(); // or notifyAll()
        }
    }
}
```

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?
3. **Check that it's our turn, wait otherwise**
4. Do work (run)
5. Update the condition
6. Notify waiting threads

```
public class WaitNotifyRunnerThread extends Thread {

    private .................. x;
    private int id;

    public WaitNotifyRunnerThread(int id, .................. x) {
        this.id = id;
        this.x = x;
    }

    public ................. void run(){
        synchronized (x) {
            while (condition) {
                x.wait();
            }
            // laufen / do running
            // update the condition
            x.notify(); // or notifyAll()
        }
    }
}
```

```
while (y.get() != this.id) {
```

original condition

can we reuse it?

Yes!

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?
3. **Check that it's our turn, wait otherwise**
4. Do work (run)
5. Update the condition
6. Notify waiting threads

```
public class WaitNotifyRunnerThread extends Thread {

    private AtomicInteger x;
    private int id;

    public WaitNotifyRunnerThread(int id, AtomicInteger x) {
        this.id = id;
        this.x = x;
    }

    public ................. void run(){
        synchronized (x) {
            while (x.get() != this.id) {
                 x.wait();
            }
            // laufen / do running
           // update the condition
          x.notify(); // or notifyAll()
        }
    }
}
```

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?
3. Check that it's our turn, wait otherwise
4. Do work (run)
5. **Update the condition**
6. Notify waiting threads

```java
public class WaitNotifyRunnerThread extends Thread {

    private AtomicInteger x;
    private int id;

    public WaitNotifyRunnerThread(int id, AtomicInteger x) {
        this.id = id;
        this.x = x;
    }

    public ................. void run(){
        synchronized (x) {
            while (x.get() != this.id) {
                x.wait();
            }
            // laufen / do running
           x.incrementAndGet(); // update the condition
           x.notify(); // or notifyAll()
        }
    }
}
```

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?
3. Check that it's our turn, wait otherwise
4. Do work (run)
5. Update the condition
6. **Notify waiting threads**

Should we use notify or notifyAll?

```
public class WaitNotifyRunnerThread extends Thread {

    private AtomicInteger x;
    private int id;

    public WaitNotifyRunnerThread(int id, AtomicInteger x) {
        this.id = id;
        this.x = x;
    }

    public ................. void run(){
        synchronized (x) {
            while (x.get() != this.id) {
                 x.wait();
            }
            // laufen / do running
           x.incrementAndGet(); // update the condition
           x.notify(); // or notifyAll()
        }
    }
}
```

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?
3. Check that it's our turn, wait otherwise
4. Do work (run)
5. Update the condition
6. **Notify waiting threads**

Should we use notify or notifyAll?

notifyAll because multiple threads can be waiting

```
public class WaitNotifyRunnerThread extends Thread {

    private AtomicInteger x;
    private int id;

    public WaitNotifyRunnerThread(int id, AtomicInteger x) {
        this.id = id;
        this.x = x;
    }

    public .................. void run(){
        synchronized (x) {
            while (x.get() != this.id) {
                 x.wait();
            }
            // laufen / do running
           x.incrementAndGet(); // update the condition
           x.notifyAll();
        }
    }
}
```

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?
3. Check that it's our turn, wait otherwise
4. Do work (run)
5. Update the condition
6. Notify waiting threads

```
public class WaitNotifyRunnerThread extends Thread {

    private AtomicInteger x;
    private int id;

    public WaitNotifyRunnerThread(int id, AtomicInteger x) {
        this.id = id;
        this.x = x;
    }

    public ................. void run(){
        synchronized (x) {
            while (x.get() != this.id) {
                x.wait();
            }
            // laufen / do running
           x.incrementAndGet(); // update the condition
           x.notifyAll();
        }
    }
}
```

Can we replace

`synchronized (x)`

with synchronizing the method?

No

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?
3. Check that it's our turn, wait otherwise
4. Do work (run)
5. Update the condition
6. Notify waiting threads

```
public class WaitNotifyRunnerThread extends Thread {

    private AtomicInteger x;
    private int id;

    public WaitNotifyRunnerThread(int id, AtomicInteger x) {
        this.id = id;
        this.x = x;
    }

    public void run(){
        synchronized (x) {
            while (x.get() != this.id) {
                 x.wait();
            }
            // laufen / do running
           x.incrementAndGet(); // update the condition
           x.notifyAll();
        }
    }
}
```

Can we replace

`AtomicInteger x`

With

`Integer x`

?

SOLA Stafette

Complete the implementation using wait/notify

1. We'll definitely need synchronization
2. What to synchronize on?
3. Check that it's our turn, wait otherwise
4. Do work (run)
5. Update the condition
6. Notify waiting threads

```
public class WaitNotifyRunnerThread extends Thread {

    private Integer x;
    private int id;

    public WaitNotifyRunnerThread(int id, Integer x) {
        this.id = id;
        this.x = x;
    }

    public void run(){
        synchronized (x) {
            while (x.get() != this.id) {
                 x.wait();
            }
            // laufen / do running
           x += 1; // update the condition
           x.notifyAll();
        }
    }
}
```

Can we replace

`AtomicInteger x`

With

`Integer x`

?

SOLA Stafette

Complete the implementation using wait/notify

Do these assumptions still hold?

Initial value of x?

0

Reference to object x?

needs to be the same object shared across all the threads

No, because x += 1 in Java creates a new Object!

```
public class WaitNotifyRunnerThread extends Thread {

    private Integer x;
    private int id;

    public WaitNotifyRunnerThread(int id, Integer x) {
        this.id = id;
        this.x = x;
    }

    public void run(){
        synchronized (x) {
            while (x.get() != this.id) {
                x.wait();
            }
            // laufen / do running
           x += 1; // update the condition
           x.notifyAll();
        }
    }
}
```

Can we replace

`AtomicInteger x`

With

`Integer x`

?

Exercise 5

Divide and Conquer

Examples of divide-and-conquer algorithms are quicksort, mergesort, Strassen matrix multiplication, and many others.

Basic structure of a divide-and-conquer algorithm:

1. If problem is small enough, solve it directly
2. Otherwise
   a. Break problem into subproblems
   b. Solve subproblems recursively
   c. Assemble solutions of subproblems into overall solution

Adding Numbers from Vector: (Recursive Version)

```
public static int do_sum_rec(int[] xs, int l, int h) {
    int size = h - l;
    if (size == 1)
          return xs[l];

    int mid  = size / 2;
    int sum1 = do_sum_rec(xs, l, l + mid);
    int sum2 = do_sum_rec(xs, l + mid, h);

    return sum1 + sum2;
}
```

Parallel Version: Task Parallelism Model

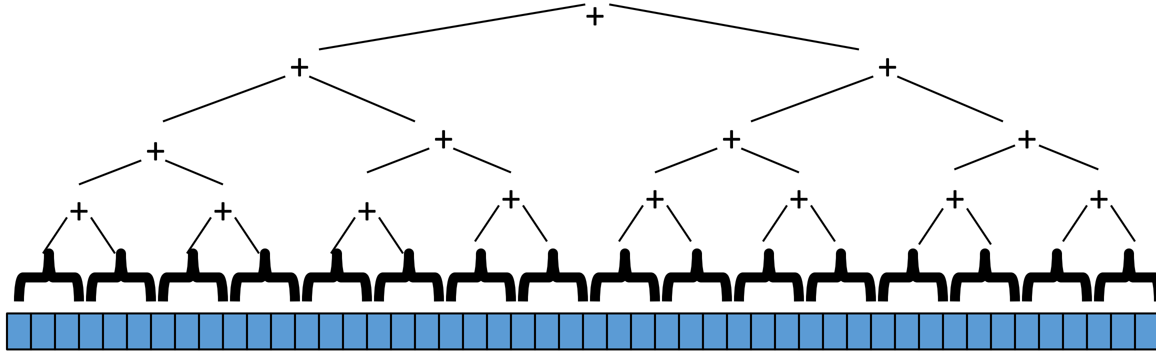
Basic flow of operations

- Create parallel tasks
- Wait for parallel tasks to complete

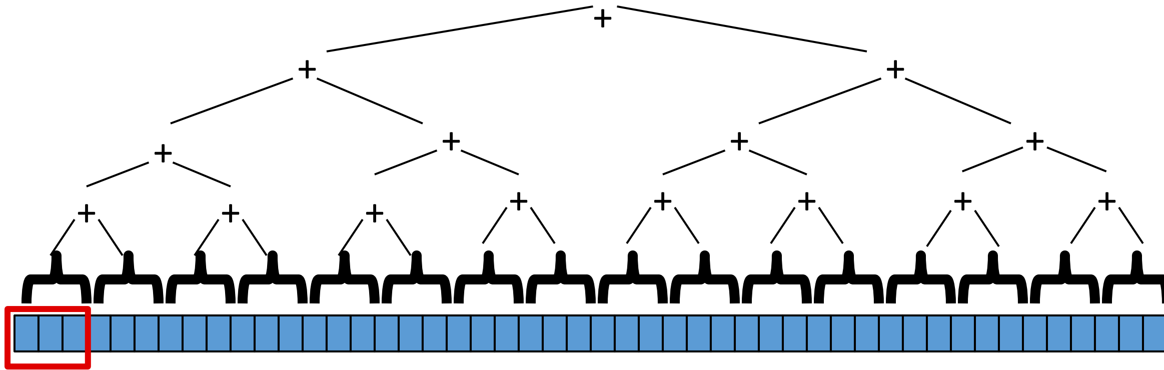
Parallel versions of divide-and-conquer on the task model are trivial:

- Create a task for the first and second part
- Wait for tasks to complete
- Combine their results

Divide and Conquer

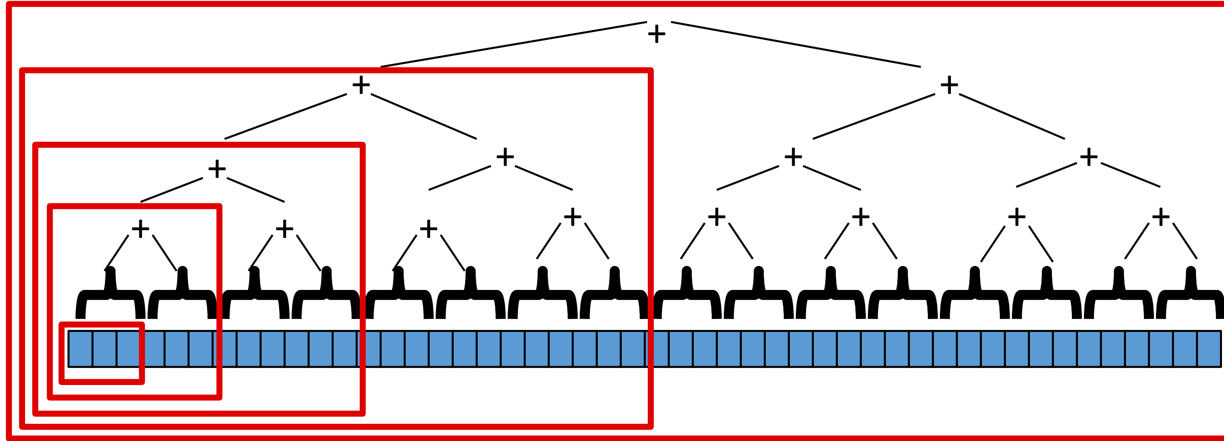


Divide and Conquer



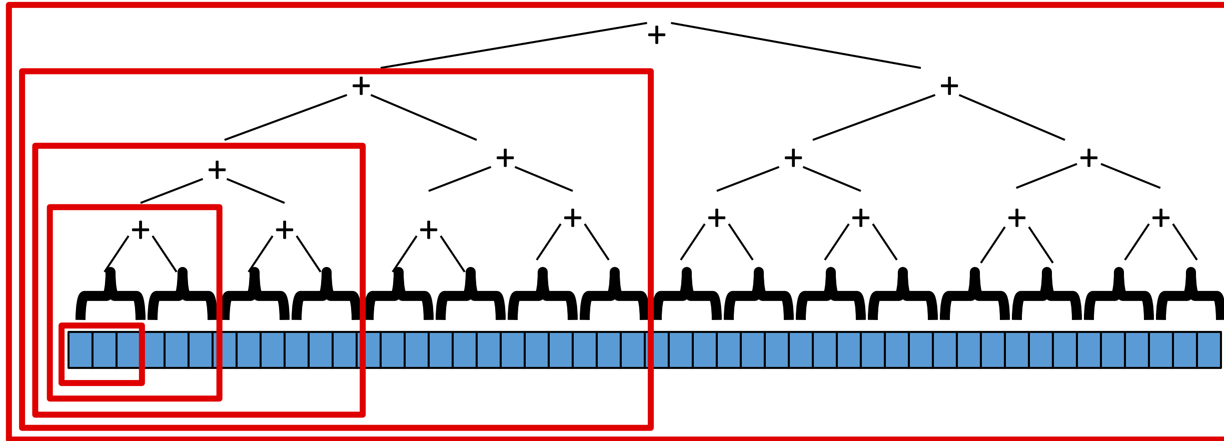
base case
no further split

Divide and Conquer



Divide and Conquer

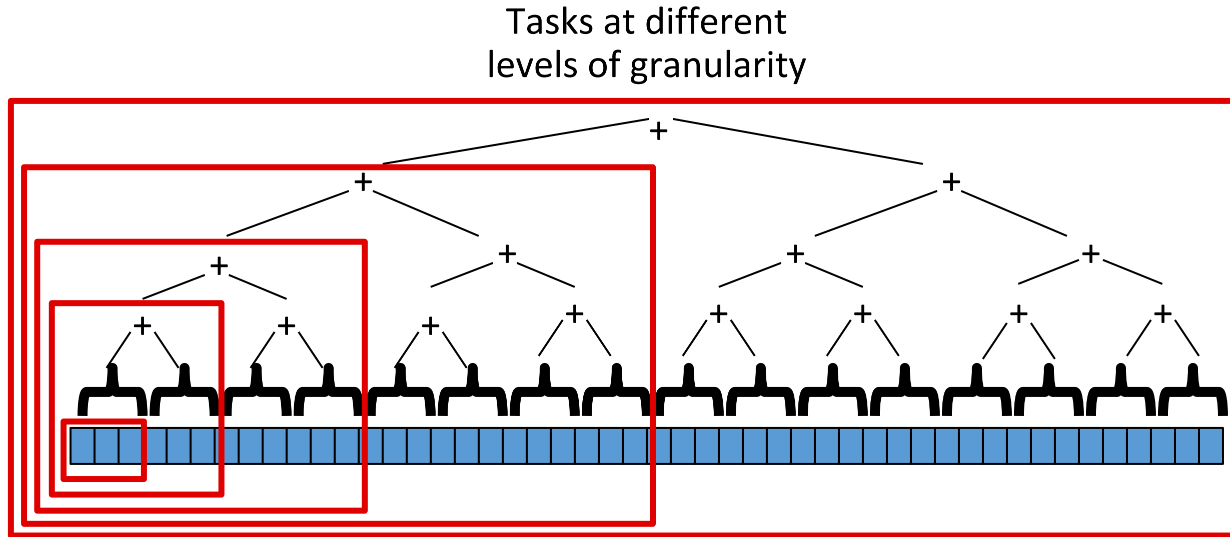
Tasks at different levels of granularity



What determines a task?

Divide and Conquer

Tasks at different
levels of granularity



What determines a task?

i) input array ii) start index iii) length/end index

These are fields we want to store in the task

Assignment 5

1. *Parallel Search and Count*

   Search an array of integers for a certain feature and count integers that have this feature.

   - Light workload: count number of non-zero values.
   - Heavy workload: count how many integers are prime numbers.

   We will study single threaded and multi-threaded implementation of the problem.

2. *Amdahl's and Gustafson's Law*
3. *Amdahl's and Gustafson's Law II*
4. *Task Graph*