

Parallelism

Bernd Burgstaller
Yonsei University

Bernhard Scholz
The University of Sydney

Outline

- Types of parallelism
 - Task-parallelism
 - Data-parallelism
 - Stream-parallelism
- Examples

Example: preparing a salad



Tasks to do:



- tear leaves
- wash leaves
- chop leaves



- wash
- slice

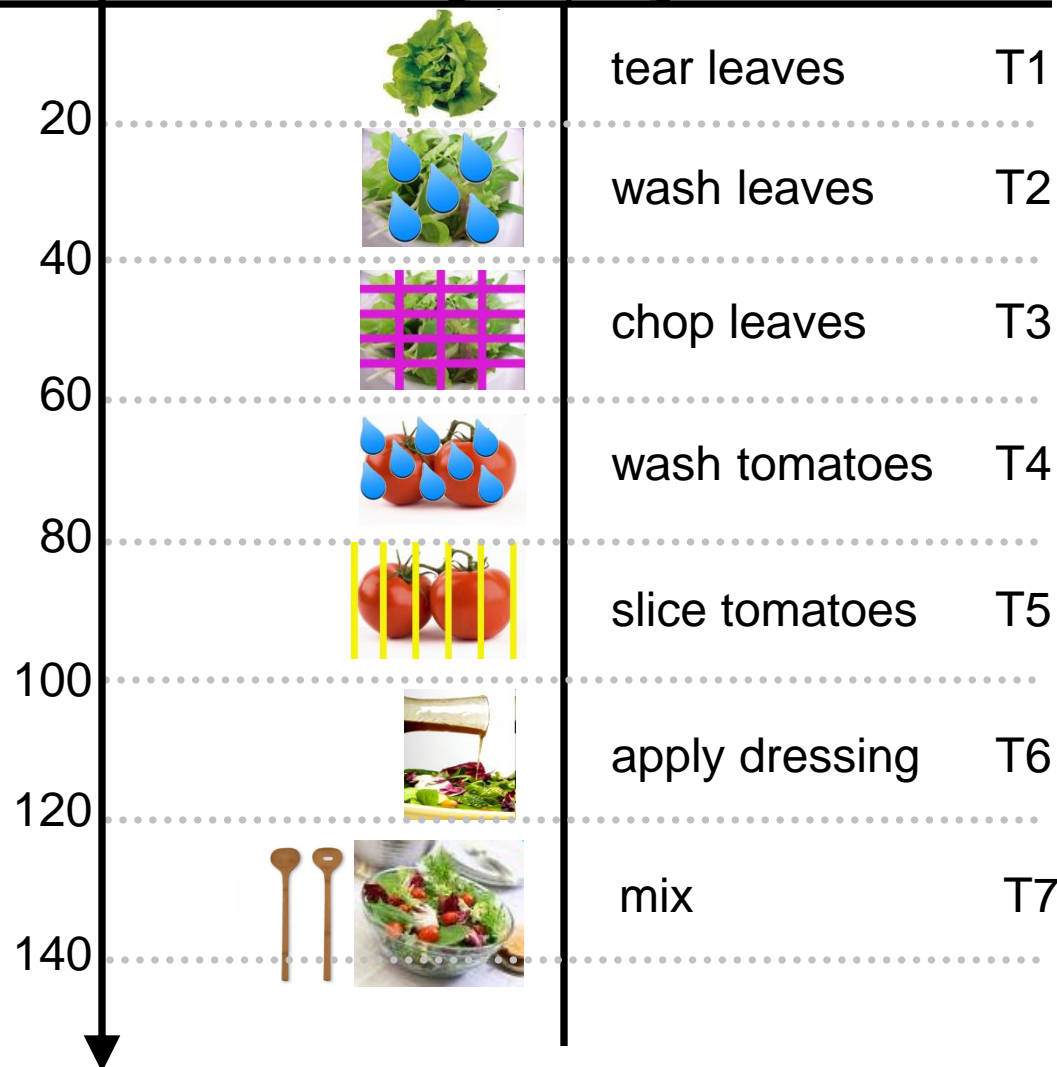


- apply dressing
- mix



Single Chef Salad (sequential)

execution time (seconds)

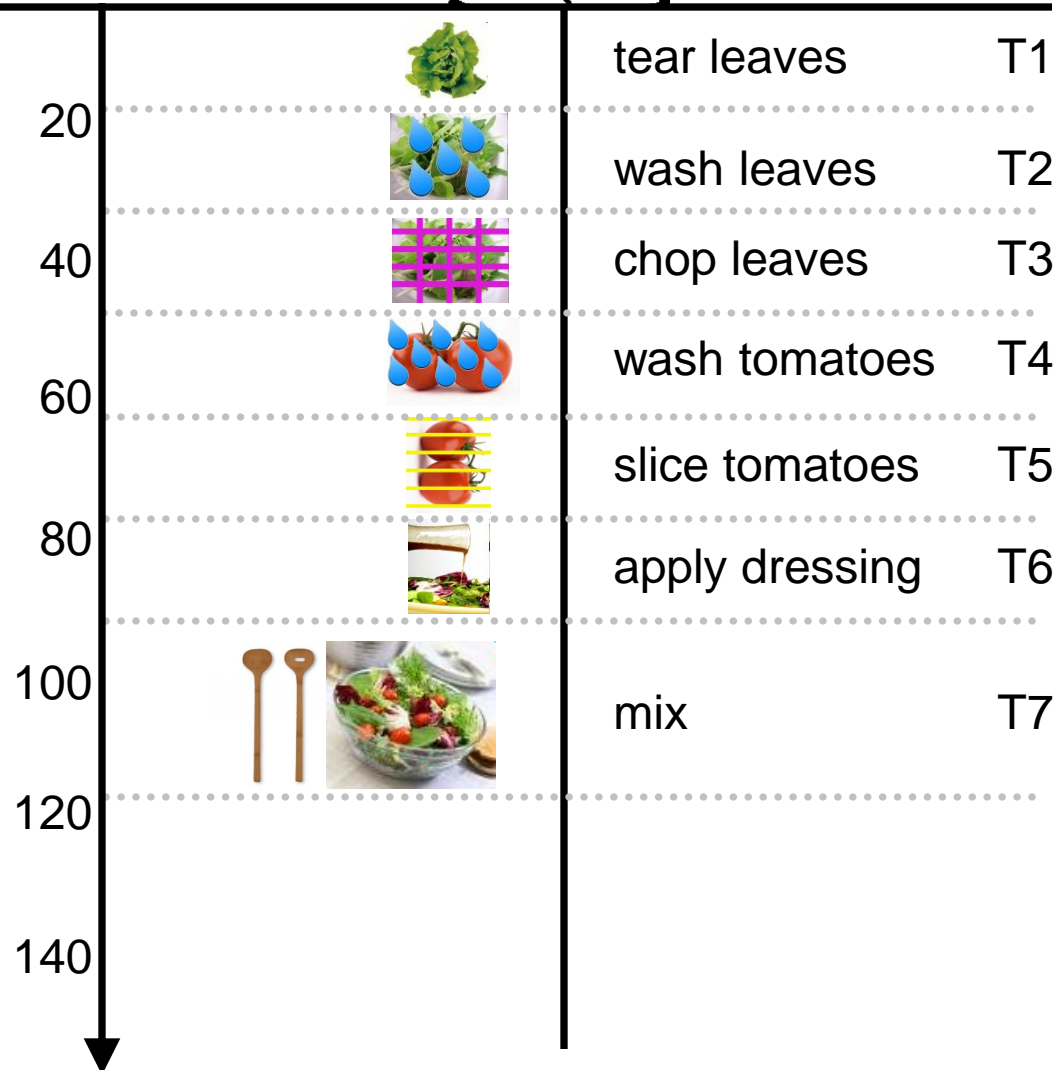


- Preparing a salad consists of 7 tasks (T1-T7).
- A single chef prepares a salad in processing the 7 tasks **sequentially** (one after the other).
- It takes a single chef 142 seconds to prepare a salad.
- We can speed up the process by making the chef work **faster**.



Fast Single Chef Salad (sequential)

execution time (seconds)



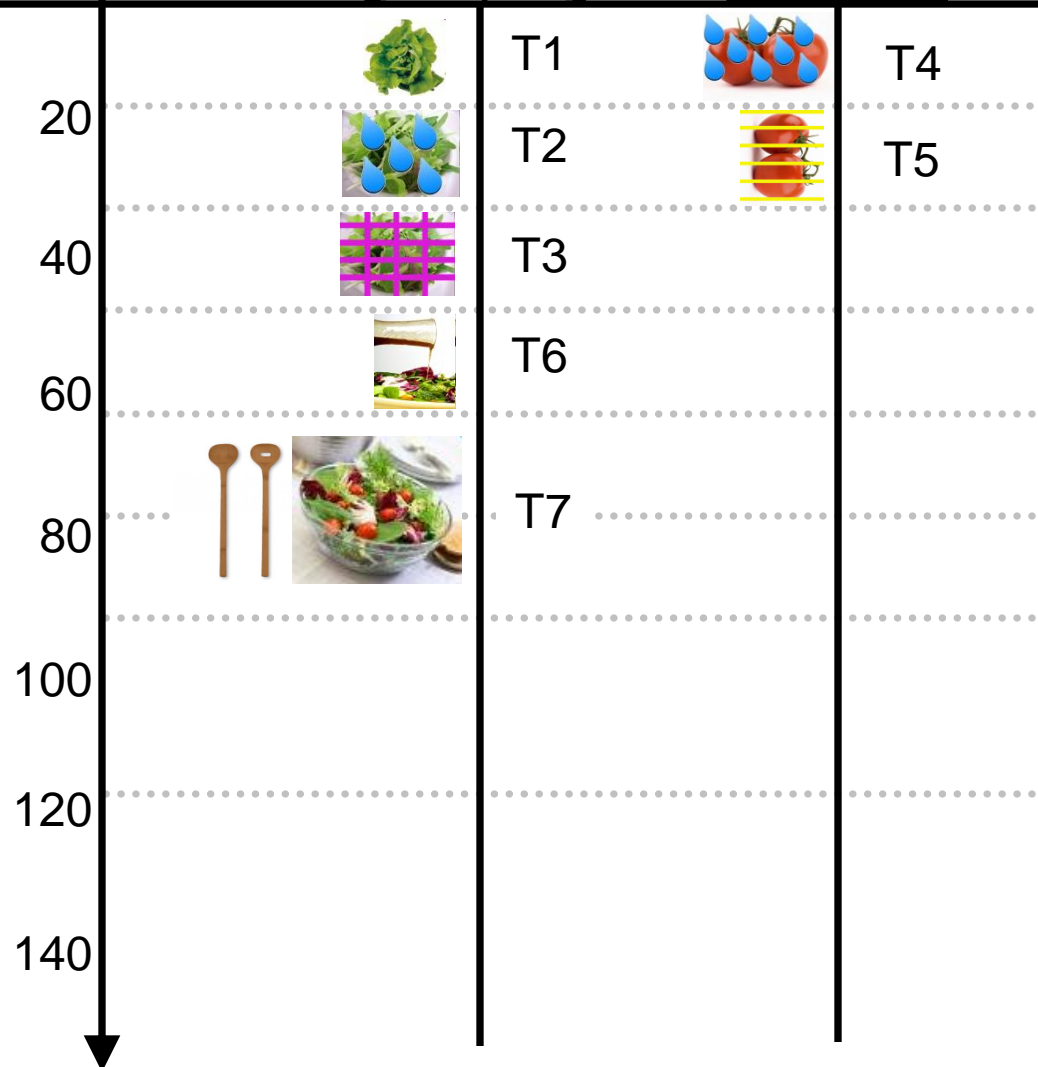
- If we can convince the Chef to finish every task in 18 instead of 20 seconds, we will reduce the process to $7 \times 18 = 126$ seconds.
- The chef agrees for Tasks T1-T6.
- However, he cannot speed up Task T7 without spoiling the entire kitchen.
- He refuses to work any faster than that.
- We can speed up the process by bringing in an additional Chef that works **in parallel** to Chef 1.



execution time
(seconds)

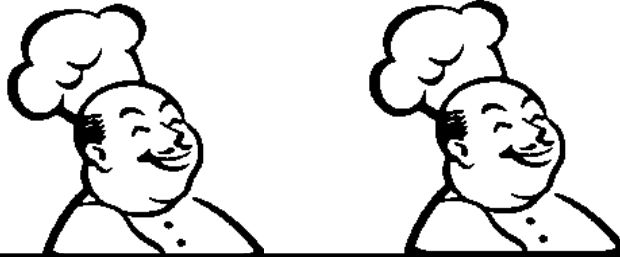


Two Chefs in Parallel Salad

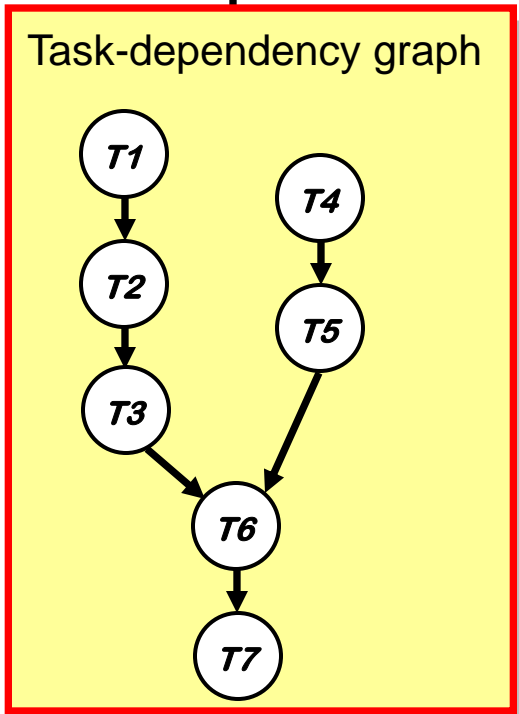
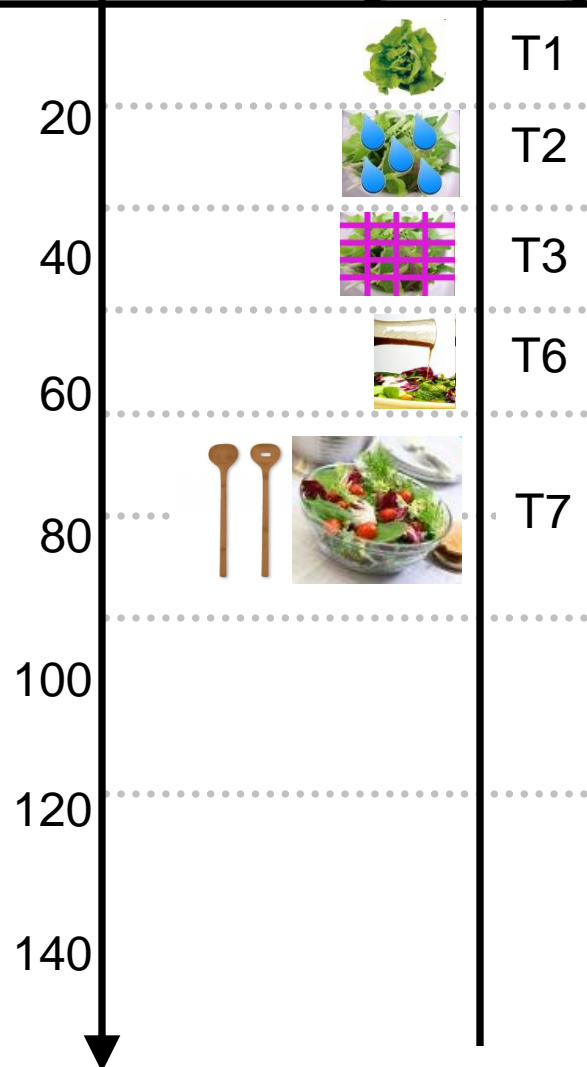


- Chef 2 washes the tomatoes (Task T4) while Chef 1 tears the lettuce (Task T1).
- Task T1 is processed in parallel to Task T4. Processing tasks in parallel is called **task-parallelism**.
- Another form of task-parallelism occurs between tasks T2 and T5.
- There exist **dependencies** between tasks:
 - Vegetables must be washed *before* they are chopped/sliced.
 - Mixing (T7) can only be done *after* the dressing has been applied (T6).

Two Chefs in Parallel Salad



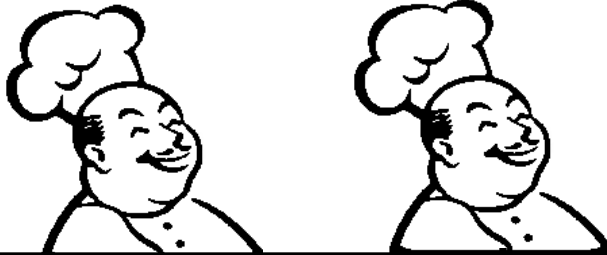
execution time (seconds)



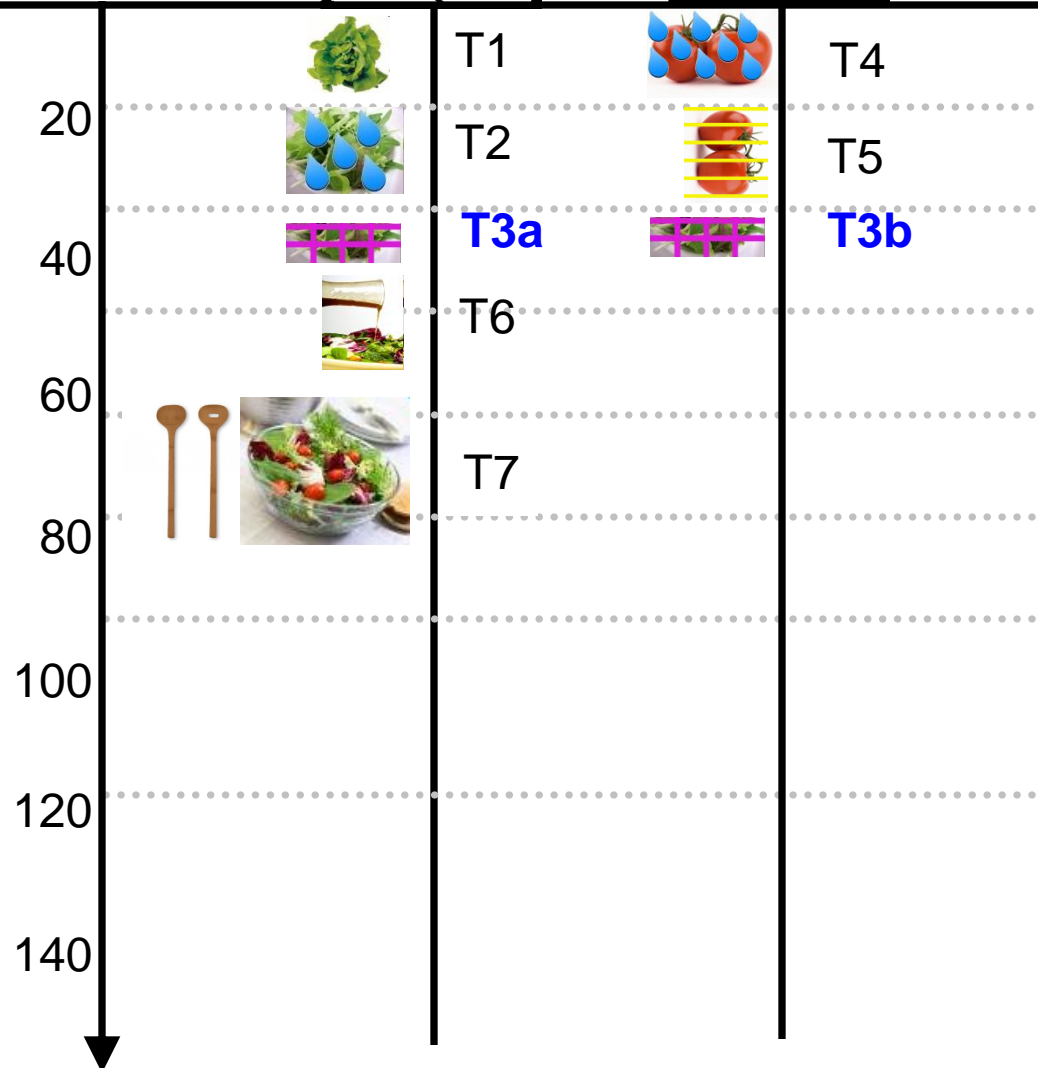
- Dependencies between tasks do not allow further task-parallelism with this example.
- A **task-dependency graph** shows dependencies between tasks.
 - directed acyclic graph (DAG)
 - graph nodes represent tasks
 - directed edge $T_a \rightarrow T_b$ indicates that T_b can only be processed after T_a has completed.
 - See example graph to the left.



execution time
(seconds)



Chefs in Parallel (task and data parallelism)

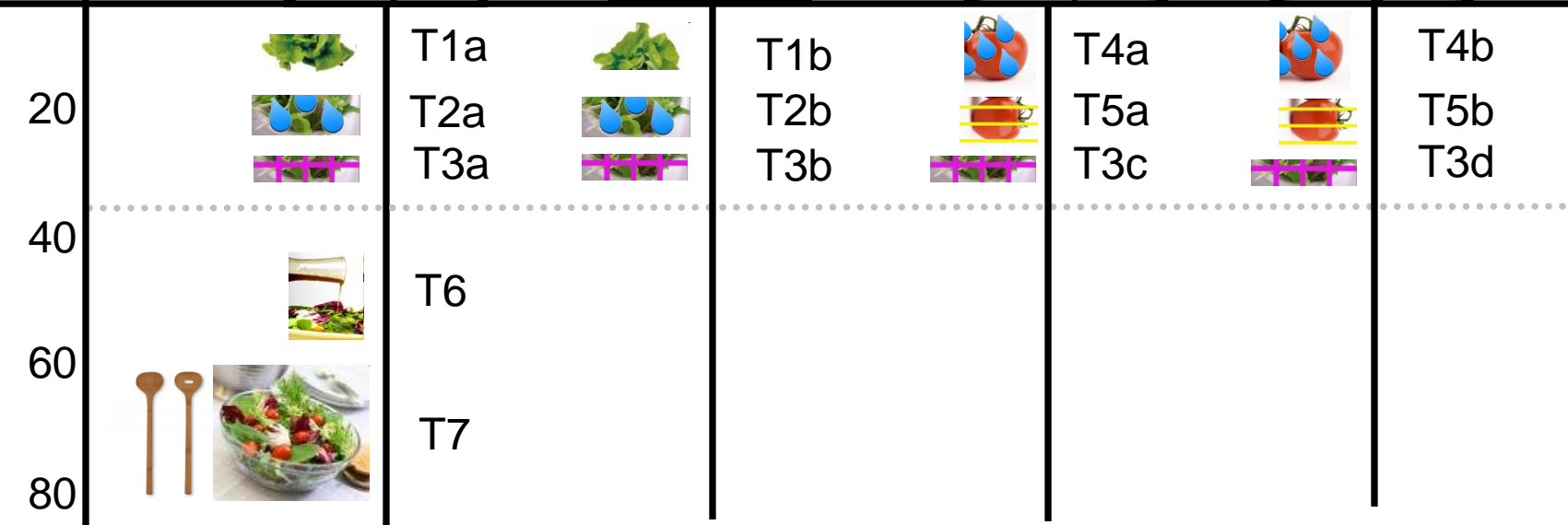


- Dependencies between tasks do not allow further task-parallelism with this example.
- However, we can have several Chefs work in parallel on the “data” of a task.
- This form of parallelism is called **data-parallelism**.
 - Example1: Chef 1 and Chef 2 chop one half of the lettuce each (Tasks **T3a** and **T3b**).
 - Example2: On the next slide more data-parallelism is introduced.

Parallelism



execution time
(seconds)



The final outcome:

- Task parallelism: T1 || T4, T2 || T5
- Data parallelism: T2a || T2b, T5a || T5b, T3a||T3b||T3c||T3d
 - We can still increase data parallelism with the lettuce by bringing in more Chefs. [What about limits?](#)

T1 || T4 means: Task T1 executes in parallel to Task T4.

Definitions

Task: a computation that consists of a sequence of instructions. The computation is a *distinct* part of a program or algorithm. (That is, programs and algorithms can be de-composed into tasks).

Examples: “washing lettuce”, “initialize data-structures”, “sort array”, ...

Task-parallelism: parallelism achieved from executing different tasks at the same time (i.e., in parallel).

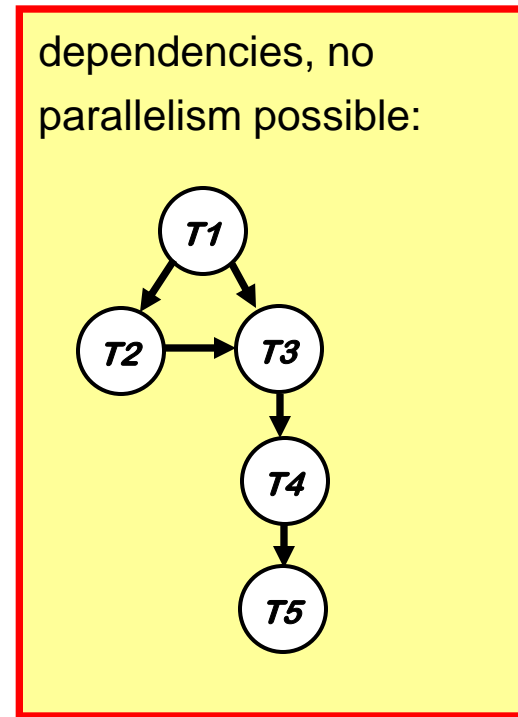
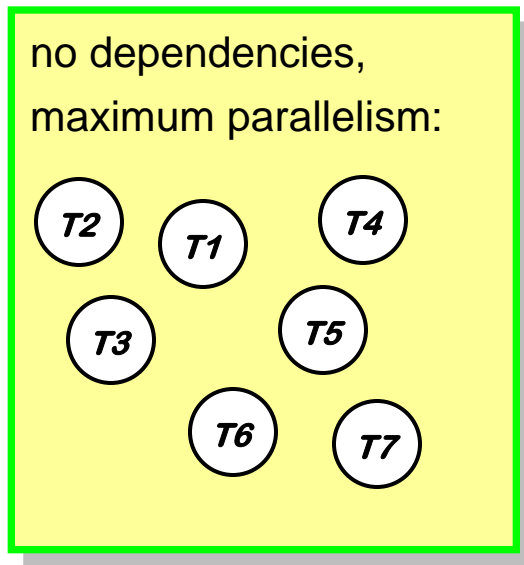
Data-parallelism: performing the same task to different data-items at the same time.

Example: 2 Chefs slicing 1 tomato each. (tomato = data, slicing ~ task).

Definitions (cont.)

Dependencies: an execution order between two tasks T_a and T_b .
 T_a must complete before T_b can execute. Notation: $T_a \rightarrow T_b$.
Dependencies limit the amount of parallelism in an application.

Example task dependency graphs:

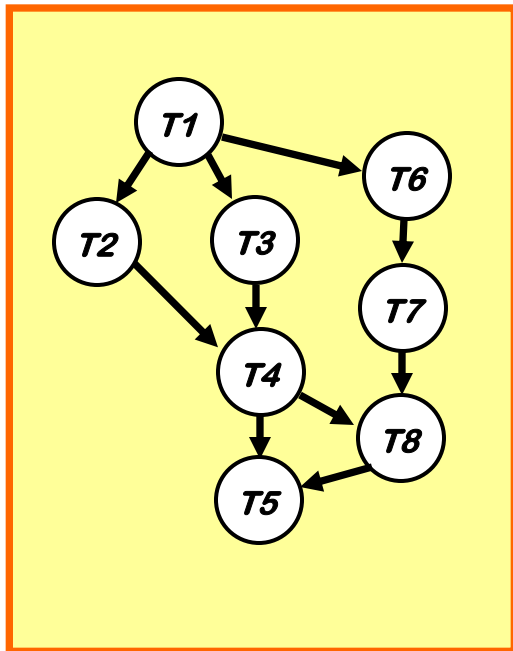


Definitions (cont. cont.)

Dependencies impose a **partial ordering** on the tasks:

Two tasks T_a and T_b can execute in parallel iff

- 1) there is no path in the dependence graph from T_a to T_b
- 2) there is no path in the dependence graph from T_b to T_a

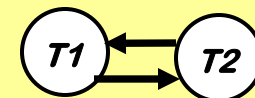


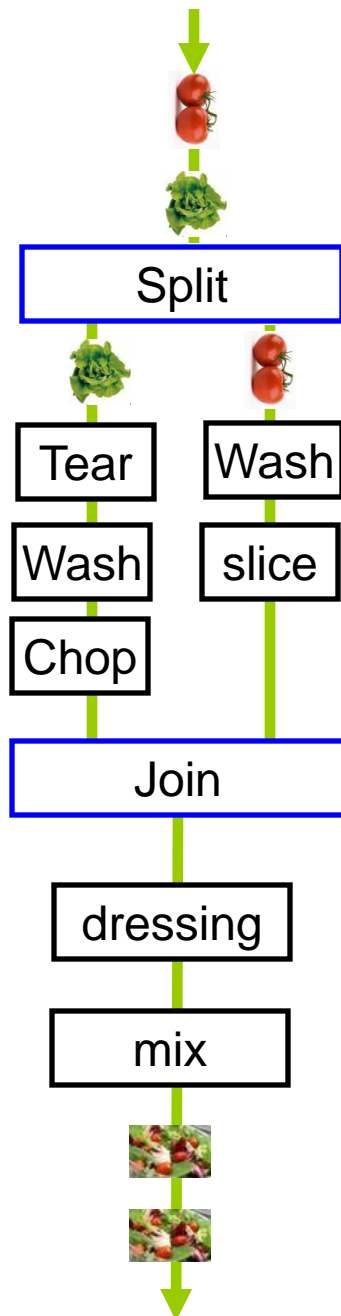
Dependency is transitive:

$T_a \rightarrow T_b$ and $T_b \rightarrow T_c$ implies $T_a \rightarrow T_c$

Say: T_a has to complete before T_b , and T_b has to complete before T_c , therefore T_a has to complete before T_c .

What about this?





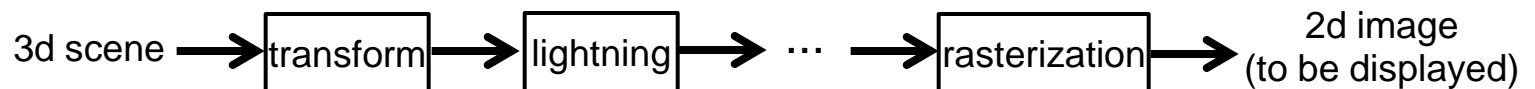
Stream Parallelism

- Some programs work on streams of data (audio, video, ...)
 - Examples: audio and video encoders/decoders, cell phone base stations.
- Tasks depicted as rectangles
 - operate in parallel -> task parallelism
 - data parallelism also easy to model (using split and join)
- “Split” divides a stream
- “Join” merges a stream
- For regular and repeating computations.
- We will discuss programming of streams with the StreamIt programming language.

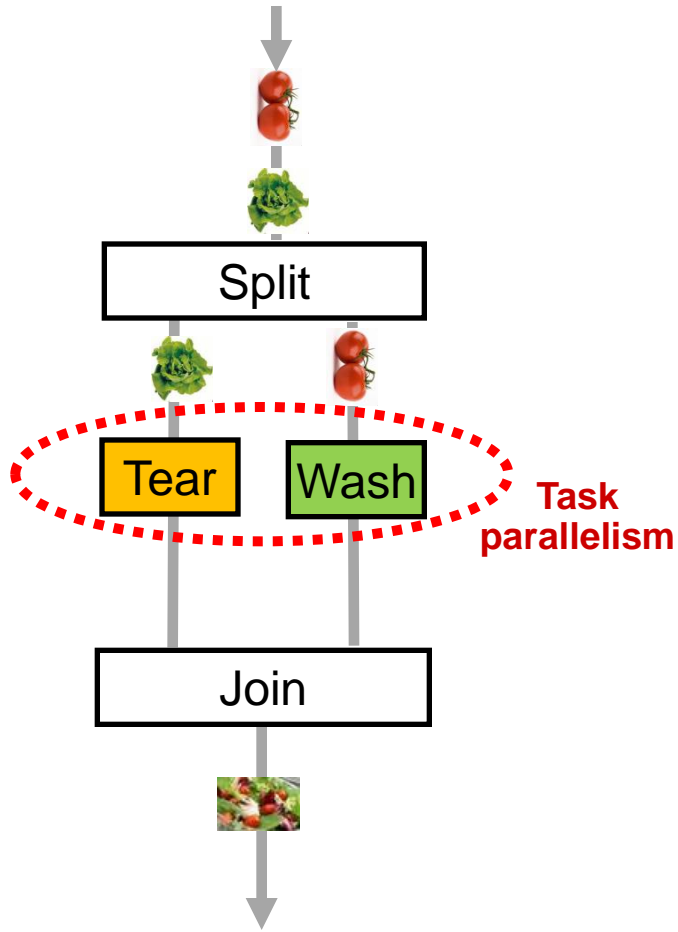
Pipeline Parallelism with Stream Programs



- Pipeline: sequence of actors
 - **producer-consumer relationship** between actors:
 - actor reads input from upstream data-channel
 - actor writes output to downstream data-channel
- Actors of pipeline operate in parallel on different data items
 - much like a factory assembly line
 - Example: while one actor tears the lettuce leaves, another actor washes torn lettuce lives that it received from the upstream actor.
- Pipeline parallelism very popular with graphics pipelines of GPUs:
 - http://en.wikipedia.org/wiki/Graphics_pipeline

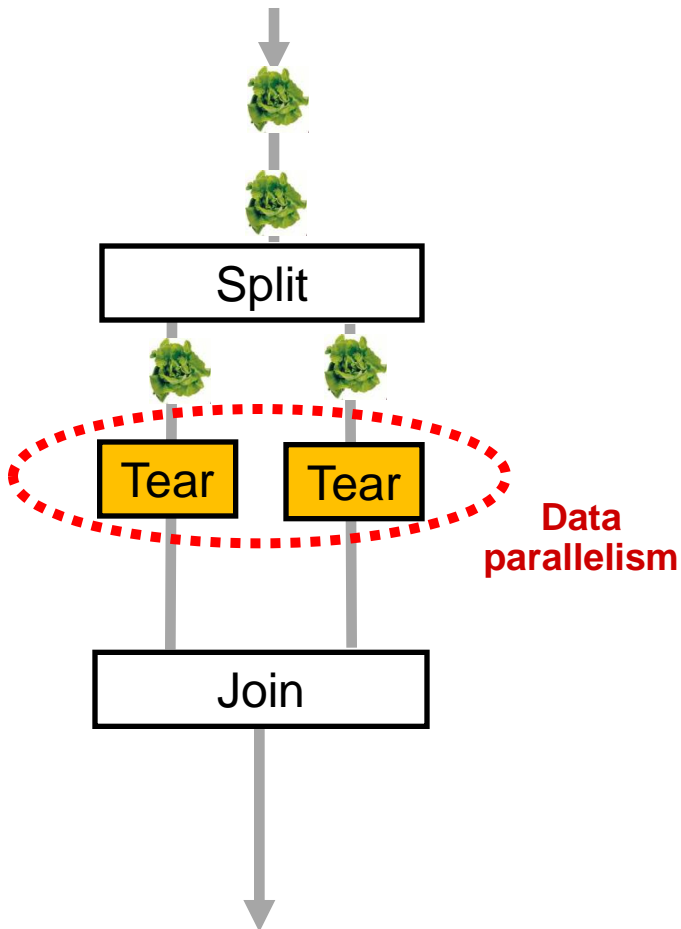


Task Parallelism with Stream Programs



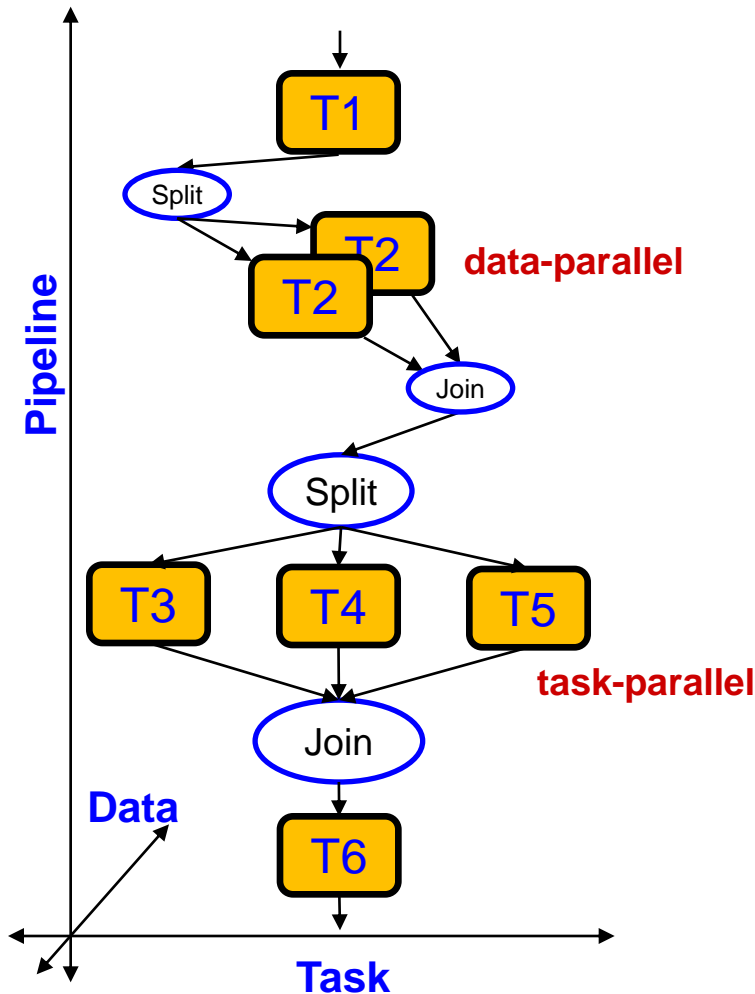
- Parallel execution of different tasks that don't have producer-consumer relationship.
- Enabled by Split and Join

Data Parallelism with Stream Programs



- Parallel execution of the same task on different data items.
- Enabled by Split and Join
 - similar to task-parallelism
- Example: in the graph on the previous slide, we can data-parallelize the lettuce 'Tear' task as shown to the left.
 - more than 2 data-parallel instances of the 'Tear' task possible!
- Not possible with stateful actors
 - a stateful actor remembers information between executions
 - Example: actor that computes the maximum over the complete data-stream
 - More on that in a latter part of the lecture...

Stream Programs: Task+Data+Pipeline Parallelism



Data Parallelism

- For *stateless* actors
- Actor duplicated within split/join pair (*actor fission*)
- Example: 2 instances of task T2

Task Parallelism

- Between actors *without* producer/consumer relationship
 - Example: T3 || T4 || T5

Pipeline Parallelism

- Between producers and consumers
- *Stateful* actors can be parallelized

Outline

- Types of parallelism ✓
 - Task-parallelism ✓
 - Data-parallelism ✓
 - Stream-parallelism ✓
- Examples

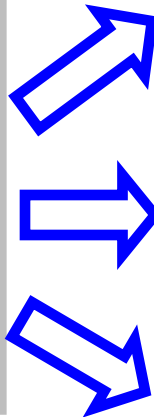
Task Parallelism Example

Example: Assume we have a large data-set in an array and the task is to compute the minimum, average and maximum value. This task can be decomposed into 3 tasks: computing minimum (T1), average (T2), and maximum value (T3).

```
#define maxN 1000000000

int m[maxN];
int i;
int min = m[0];
int max = m[0];
double avrg = m[0];

for(i=1; i < maxN; i++) {
    if(m[i] < min)
        min = m[i];
    avrg = avrg + m[i];
    if(m[i] > max)
        max = m[i];
}
avrg = avrg / maxN;
```



```
#define maxN 1000000000
int m[maxN];

int i; int min = m[0];
for(i=1; i < maxN; i++) {
    if(m[i] < min)
        min = m[i];
}
}

int j;
double avrg = m[0];
for(j=1; j < maxN; j++) {
    avrg = avrg + m[j];
}
avrg = avrg / maxN;

int k; int max = m[0];
for(k=1; k < maxN; k++) {
    if(m[k] > max)
        max = m[k];
}
}
```

T1

T2

T3

Dependence graph:



Note: if T1 – T3 should execute in parallel, then each task needs its own loop index variable (i, j, k)!

Task Parallelism Example (cont.)

```
#define maxN 1000000000
int m[maxN];
```

```
int i; int min = m[0];
for(i=1; i < maxN; i++) {
    if(m[i] < min)
        min = m[i];
}
```

T1

```
int j;
double avrg = m[0];
for(j=1; j < maxN; j++) {
    avrg = avrg + m[j];
}
avrg = avrg / maxN;
```

T2

```
int k; int max = m[0];
for(k=1; k < maxN; k++) {
    if(m[k] > max)
        max = m[k];
}
```

T3

- The problem is now decomposed into three tasks T1, T2, T3.
- However: still sequential
 - T1, then T2 then T3
- Need a way to tell the compiler that tasks T1, T2 and T3 shall be executed in parallel.
- We will use **POSIX threads** to do that.
 - Our next lecture will be on POSIX threads.
- We will discuss other ways to express task parallelism in later parts of the lecture.

Data Parallelism Example

Example 1: parallel sum computation on array (Lecture L00).

Example 2: vector operations:

```
int a[4] = {1,2,3,4};
int b[4] = {1,2,3,4};
int c[4];
int i;

for(i=0; i < 4; i++) {
    c[i] = a[i] + b[i];
}
```

- Assume two arrays of integers, compute the pair-wise sum.
- A sequential version uses a for-loop to compute the sum.
 - Requires one loop iteration per array index.
 - 4 add operations in total
- Using SSE on the Intel IA32 architecture, this sum operation can be computed in one step.
 - See example on next slides.
 - **The Intel AVX instructions of the new Sandy Bridge architecture supports vectors of 8 ints.**

SIMD Vectorization

- To sum the values of 2 arrays, a conventional CPU needs one add operation (“+”) per array index:

```
int a[4] = {1,2,3,4};
int b[4] = {1,2,3,4};
int c[4];
```

```
c[0] = a[0] + b[0];
c[1] = a[1] + b[1];
c[2] = a[2] + b[2];
c[3] = a[3] + b[3];
```

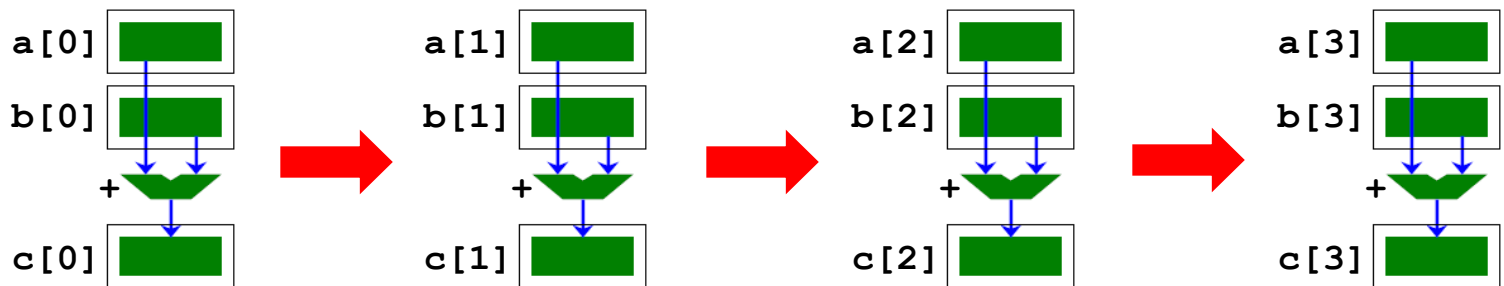
sequential array sum

```
int a[4] = {1,2,3,4};
int b[4] = {1,2,3,4};
int c[4];
int i;
```

```
for(i=0; i < 4; i++) {
    c[i] = a[i] + b[i];
}
```

array sum using loop

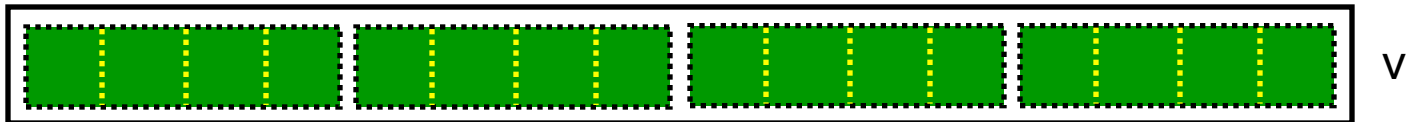
- Reason: a register of a conventional CPU can only hold only 1 data item at a time (such a register is called a **scalar register**):



Data Parallelism Example (cont.)

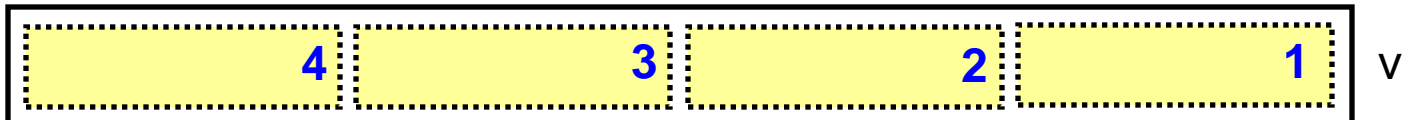
- Vector processors have large registers that can hold *multiple* values of the *same* data type (vector register).
- The SSE registers of the Intel x86 are 128 bit wide.

- `__m128i v;` declares vector `v`, which consists of four 4-byte integers:



- The `__m128i` vector data type is provided in header file `emmintrin.h`.
- Other provided types: 64-bit int, 64-bit float, 32-bit float, 16-bit int, 8-bit int.

`__m128i v = _mm_set_epi32(1, 2, 3, 4);` assigns values to the **elements** of `v`.

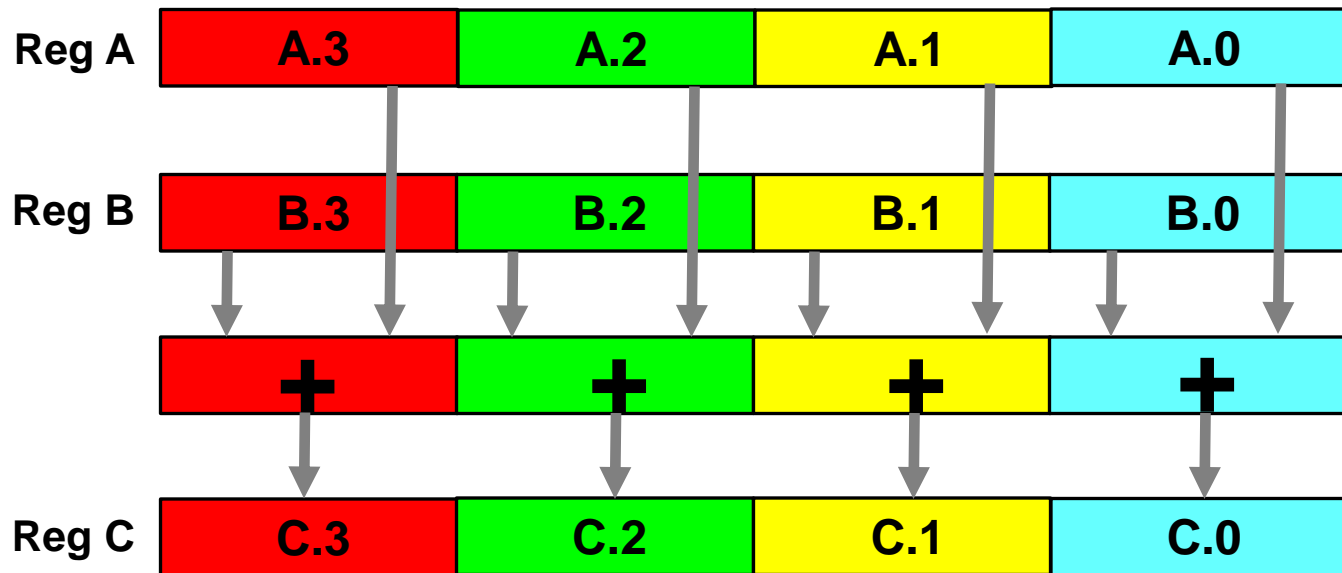


- SSE instructions will apply an operation to all elements of a vector **at once**.
 - See example on next slide.

Data Parallelism Example (cont.)

- Example: adding two vectors of integers in one step.

```
__m128i A, B, C; // declare 3 vectors A, B, C of type __m128i  
C = _mm_add_epi32(A, B);
```



Data Parallelism Example (cont.)

```
#include <smmintrin.h>
#include <emmintrin.h>
#include <stdio.h>

int main(void) {

    __m128i A = _mm_set_epi32(4, 3, 2, 1);
    __m128i B = _mm_set_epi32(2, 2, 2, 2);
    __m128i C = _mm_add_epi32(A, B); //add vectors A and B

    int res = _mm_extract_epi32(C, 3); //extract element
                                        //at index 3

    printf("Result: %d\n", res); // ..... ?

    return 0;
}
```

Note: This program is available in YSCEC.
Log in on elc1, compile & run:

```
$ gcc -msse4.1 sse_int.c
```

Data Parallelism Example (cont.)

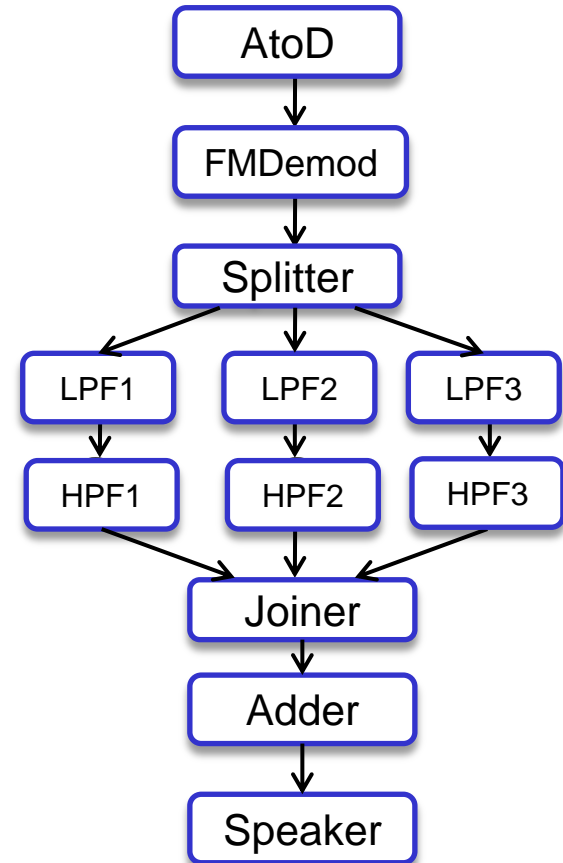
- The header files `smmintrin.h` and `emmintrin.h` contain the C language extensions for the example's SSE parts.
 - Must be included to use `_mm_add_epi32()` aso.
 - Further header files exist, to cover various SSE generations:
 - for SSE, SSE2, SSSE3, SSE4_1, SSE4_2, AVX
- `_mm_add_epi32()` is called an **intrinsic**. An intrinsic is a procedure provided by the compiler. The compiler will replace the intrinsic with one or more assembly instructions.
- Note:
 - Intel issued successive generations of SSE extensions with their line of x86 processors: SSE, SSE2, SSSE3, SSE4_1, SSE4_2, AVX
 - Older processors will only support the earlier versions (SSE, SSE2, ...)
 - Running machine-code with SSE instructions on CPU that does not support it will result in an 'illegal instruction' exception followed by program termination.
 - Sometimes SSE is also referred to as MMX (multimedia extensions).
 - MMX was the name of the first generation of SSE extensions.

Data Parallelism Example (cont.)

- Many of today's mainstream CPU architectures support vector instructions.
 - Architecture-specific
 - Intel x86: SSE and AVX extensions
 - AMD: 3DNow! for floating-point numbers
 - PowerPC: AltiVec

Stream Parallelism Example

- We will discuss StreamIT
- For programs based on streams of data
 - Audio, video, DSP, networking, cryptographic processing kernels
 - Examples: HDTV editing, radar tracking, cell phone base stations, computer graphics
- Properties of streams:
 - Regular and repeating computation
 - Independent filters (aka 'actors') with explicit communication
 - Task, data, and stream parallelism can be expressed



Outline

- Forms of parallelism ✓
 - Task-parallelism ✓
 - Data-parallelism ✓
 - Stream-parallelism ✓
- Examples ✓