CSE 291: FPGA for Computer Vision

Janarbek Matai

Department of Computer Science and Engineering

University of California, San Diego

04/06/2017

UCSDCSE
Computer Science and Engineering

Announcements

- CSE 3219 lab
  - M/W at 6: 30 to 8:00 pm
  - Vivado HLS
  - SDSoC
  - Vivado
- Assignment submission / announcements
  - Piazza
  - Bitbukcet repo: Share with Janarbek Matai and Ali Irturk
    - Must include team members bio
    - Last_name1_lastname2_cse291
      - Assignment1, Assignment2, Assignment3, Final_Project
- Assignment 1 due next 04/13/2017
- HLS Introduction
  - Overview, area/performance optimization, Vivado HLS

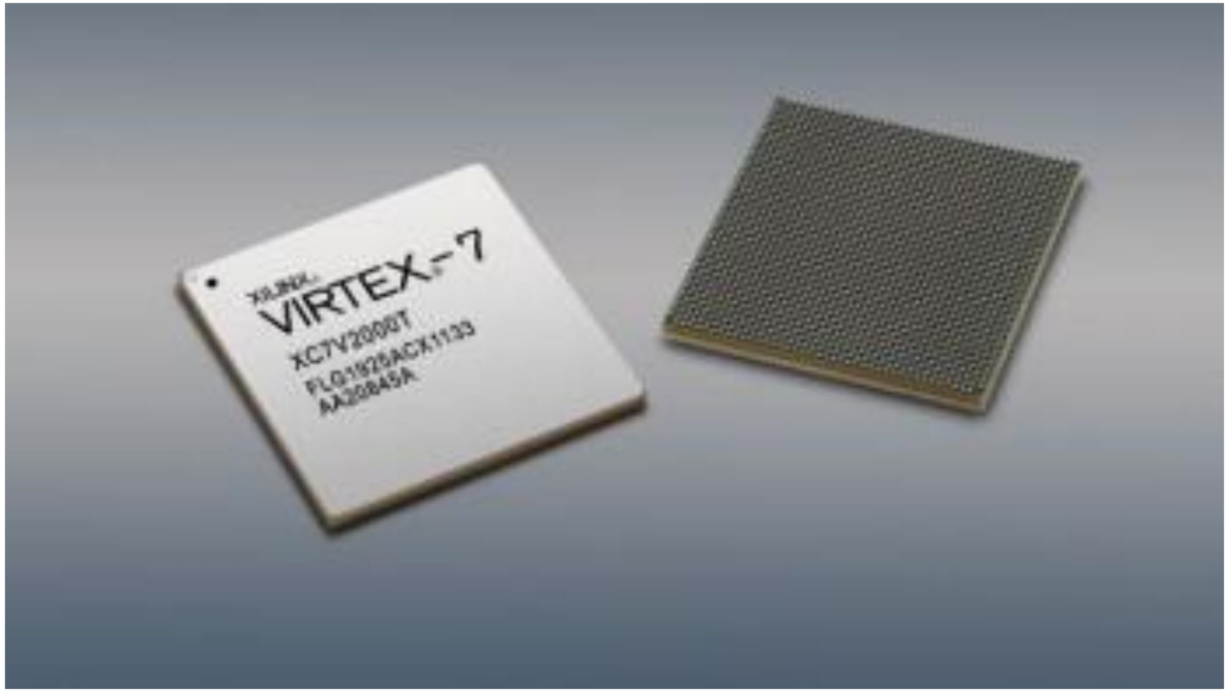
High-Level Synthesis in CSE 291

- ✓ HLS Introduction
  - ✓ Design flow with HLS
- ✓ HLS Theory (Scheduling, Binding, Allocation)
- ✓ Vivado HLS
  - ✓ Vivado HLS performance optimization
  - ✓ Vivado HLS area optimization
  - ✓ Understanding Vivado HLS report
  - FPGA
- Linear Algebra Kernel Optimizations with Vivado HLS

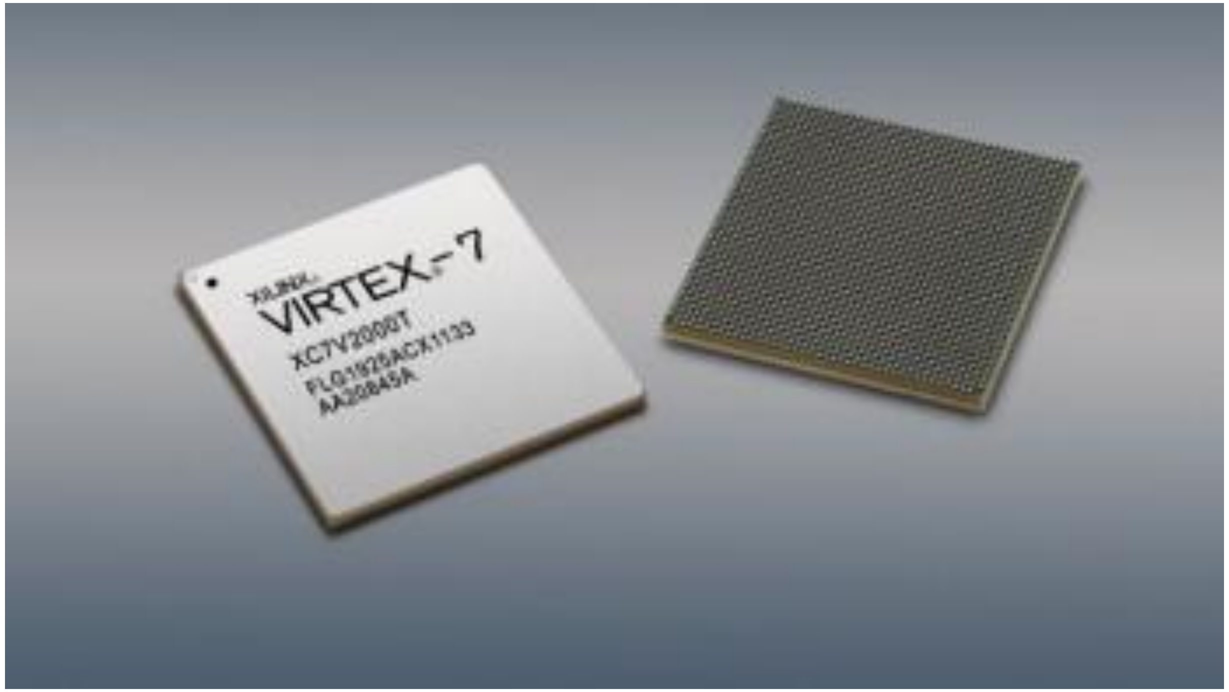
High-Level Synthesis in CSE 291 – con’t

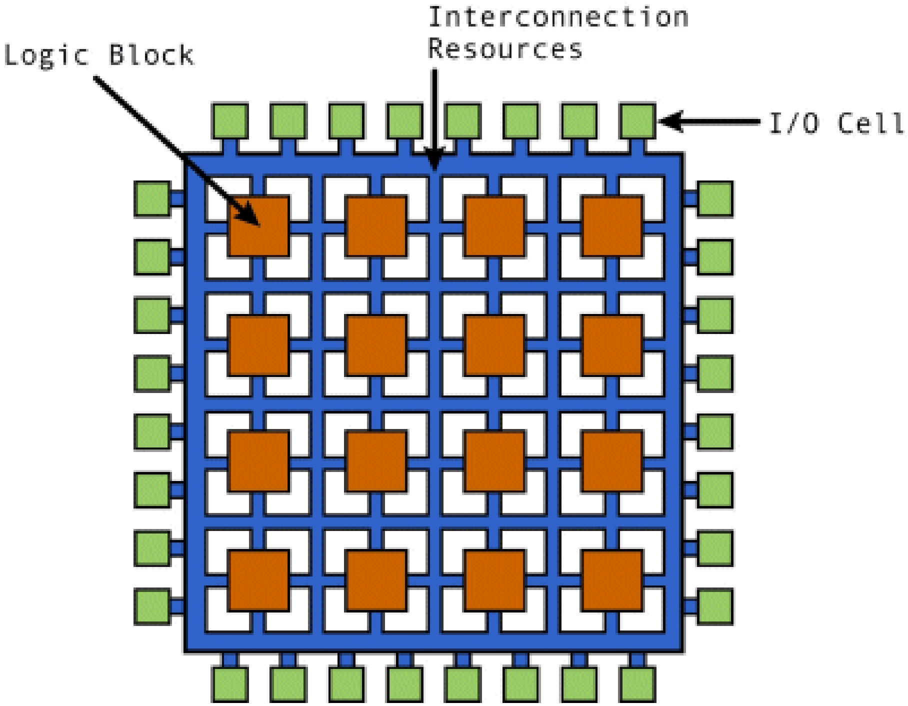
- ✓Advanced Optimizations
- ✓Restructured Code
- ✓Streaming Core Design
- ✓Applications: Face Detection, Face Recognition, CNN, BNN,…
- ✓Demo using Zedboard.

FPGA (Field Programmable Gate Arrays)



FPGA (Field Programmable Gate Arrays)



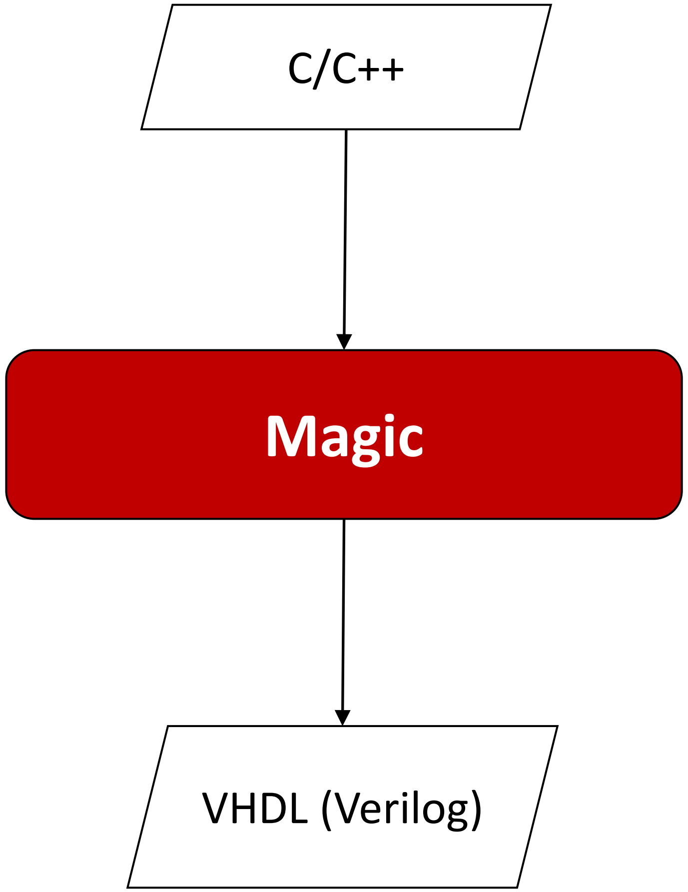


VHDL ???

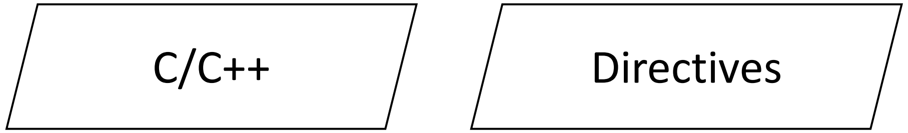
```vhdl
process (up_down, cnt) begin
    if ((up_down = '1' and cnt = 1) or
        (up_down = '0' and (cnt(WIDTH-1) = '1' and
         cnt(WIDTH-2 downto 0) = 0))) then
        overflow <= '1';
    else
        overflow <= '0';
    end if;
end process;
```

```vhdl
process (clk, reset, cnt, enable, up_down)
    variable temp_a :std_logic_vector (WIDTH-1 downto 0);
    variable temp_b :std_logic :='1';
begin

    temp_a := cnt and "01100011";
    temp_b :='1';
    for i in 0 to WIDTH-1 loop
        temp_b := temp_a(i) xnor temp_b;
    end loop;

    if (rising_edge(clk)) then
        if (reset = '1') then
            cnt <= (others=>'0');
        elsif (enable = '1') then
            if (up_down = '1') then
                cnt <= (temp_b & cnt(WIDTH-1 downto 1));
            else
                cnt <= (cnt(WIDTH-2 downto 0) & temp_b);
            end if;
        end if;
    end if;
end process;
count <= cnt;
```

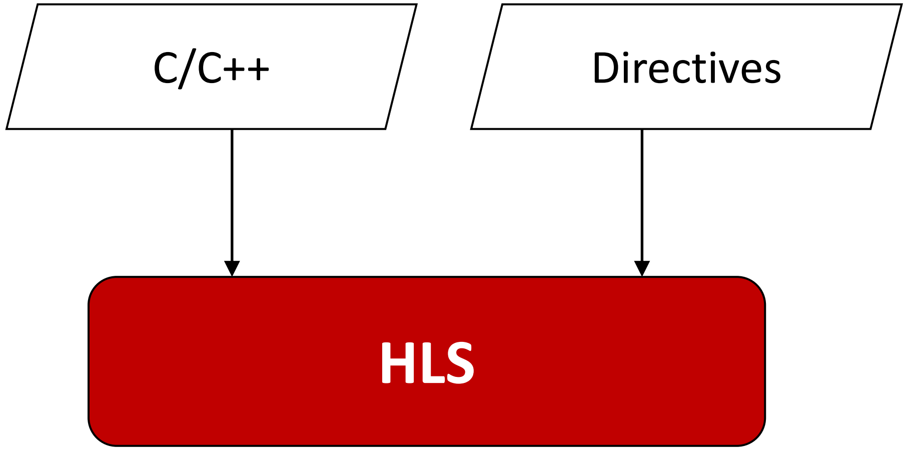
High-Level Synthesis



High-Level Synthesis

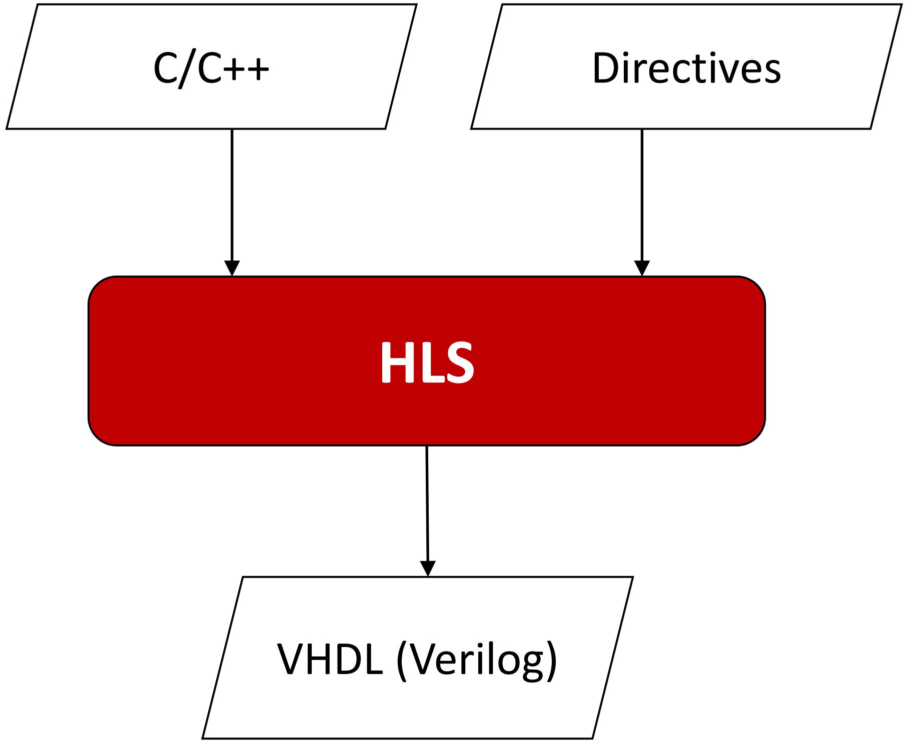


High-Level Synthesis



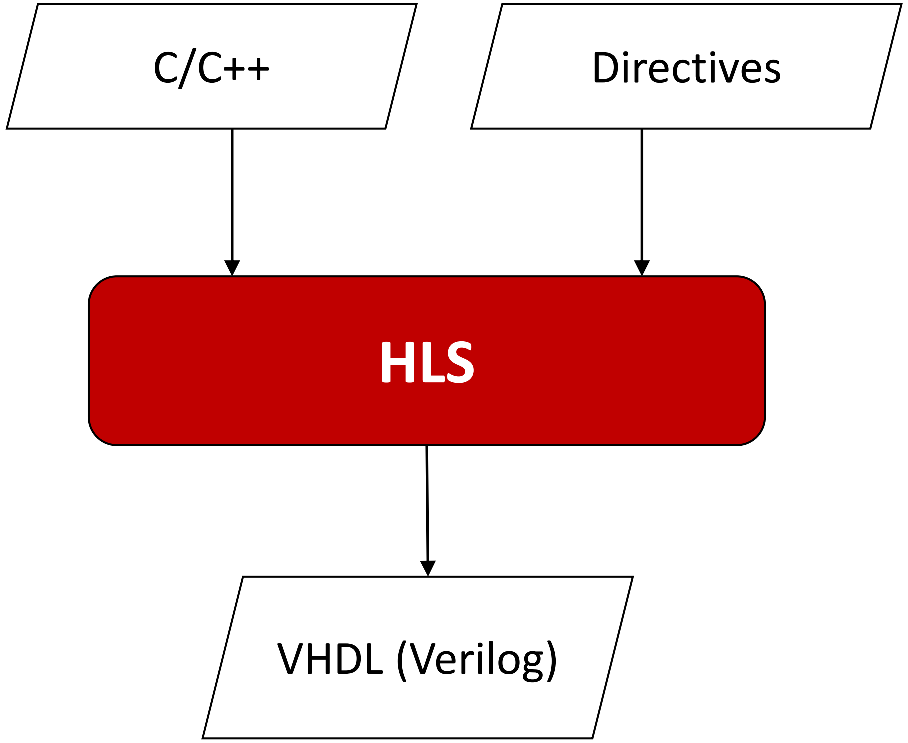
- Directives: **Pipeline, partition, dataflow,** inline,reshape, resource, stream, top, interface, expression balance, reset, allocation, dependence, function_initiate,…

High-Level Synthesis



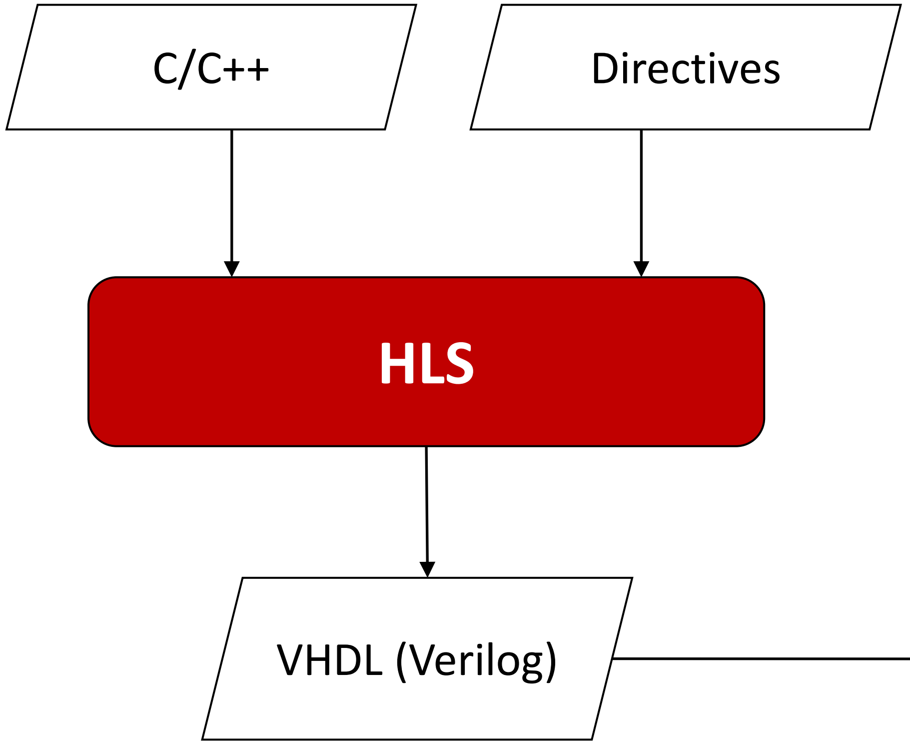
- Directives: **Pipeline, partition, dataflow,** inline,reshape, resource, stream, top, interface, expression balance, reset, allocation, dependence, function_initiate,…

High-Level Synthesis

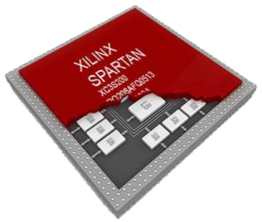



```
void HelloWorld(
   int A[8],
   int B[8]) {
   #pragma unroll
   for(int i=0;i<8;i++){
      B[i]=A[i]*3;
   }
}
```

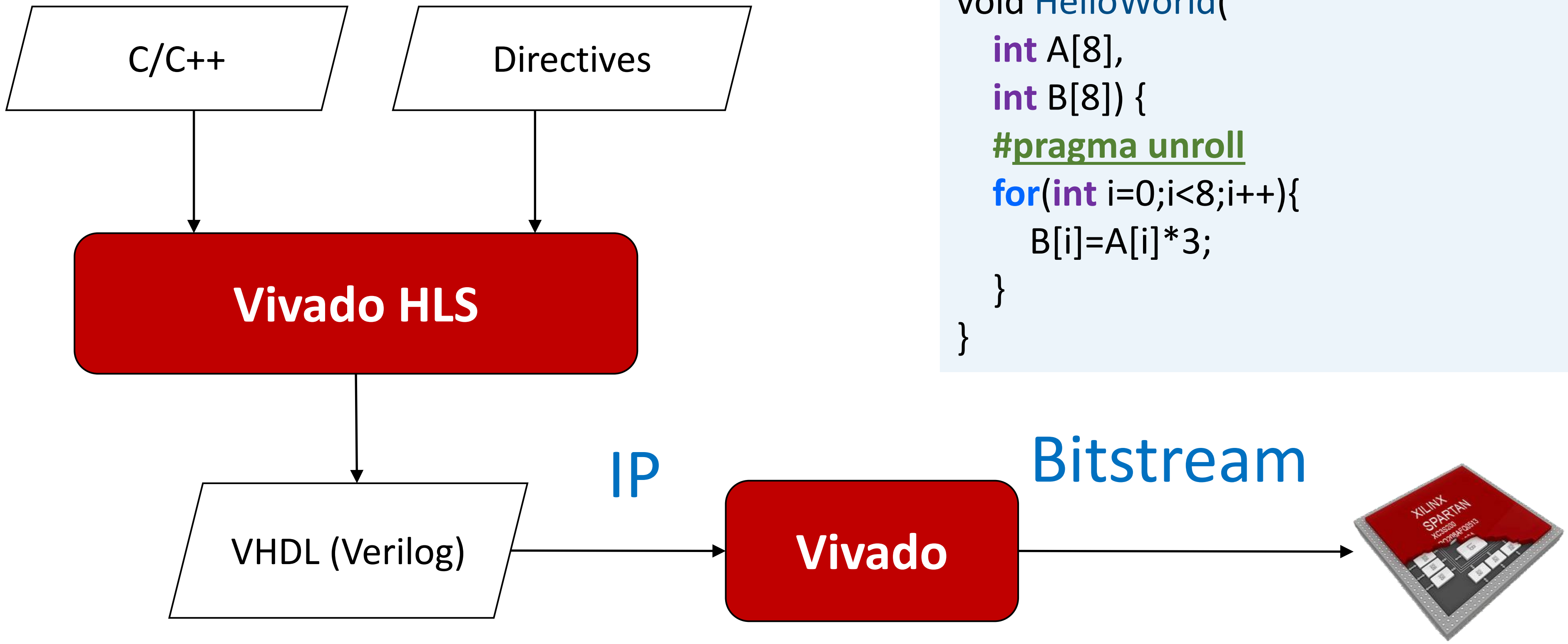
High-Level Synthesis




```
void HelloWorld(
   int A[8],
   int B[8]) {
   #pragma unroll
   for(int i=0;i<8;i++){
      B[i]=A[i]*3;
   }
}
```

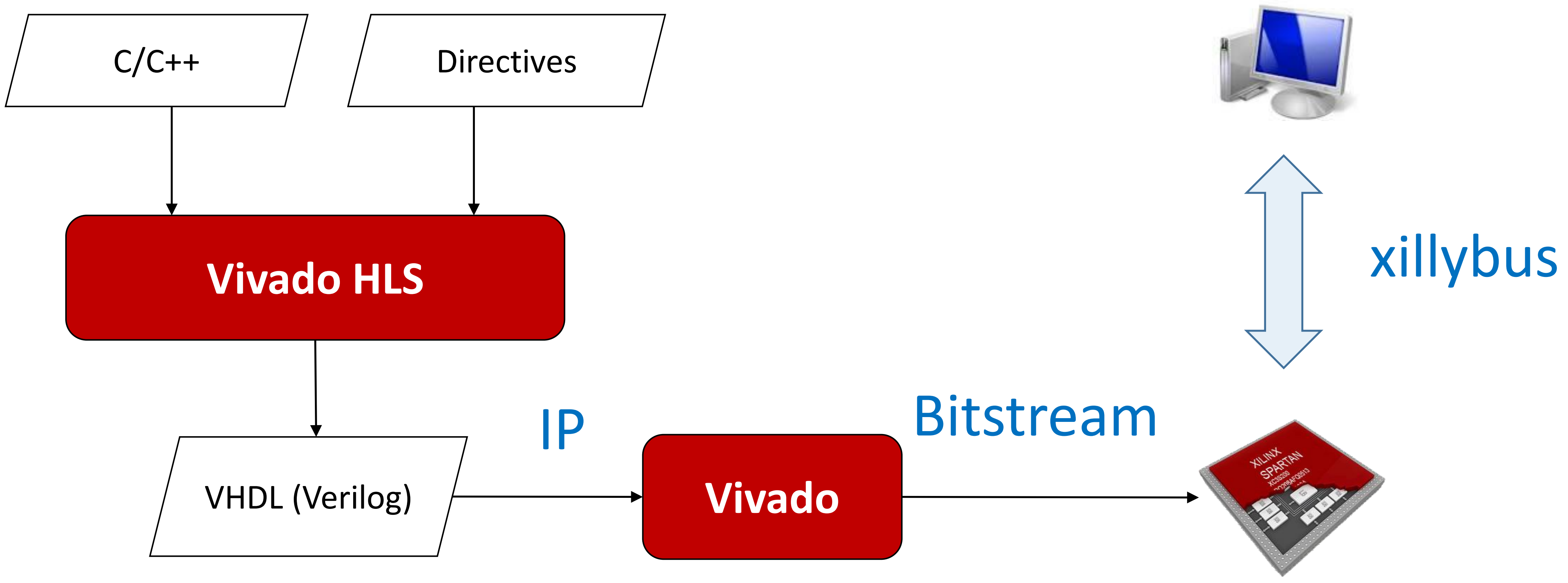


High-Level Synthesis




```
void HelloWorld(
   int A[8],
   int B[8]) {
   #pragma unroll
   for(int i=0;i<8;i++){
      B[i]=A[i]*3;
   }
}
```

High-Level Synthesis



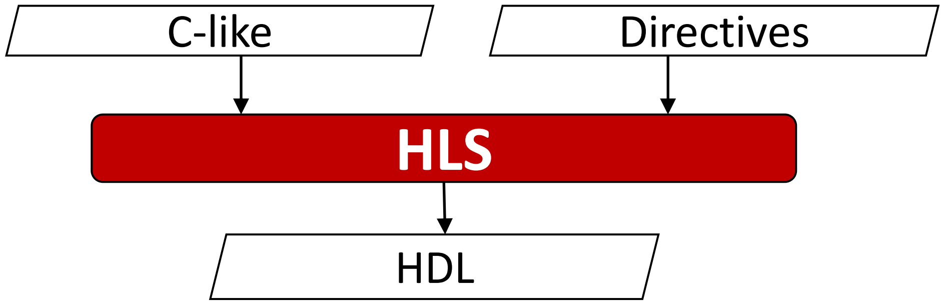
VHDL vs HLS ?

Manual design

HDL (Verilog/VHDL)

- ✓ **Pros**
  - **Good QoR**
- ✓ **Cons**
  - **Expensive**
  - **Error Prone**
  - **Debugging is difficult**

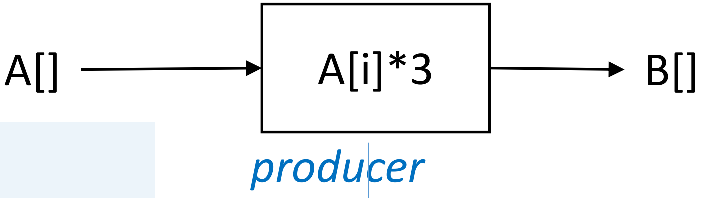
High-Level Synthesis (HLS)



Directives: Pipeline,…

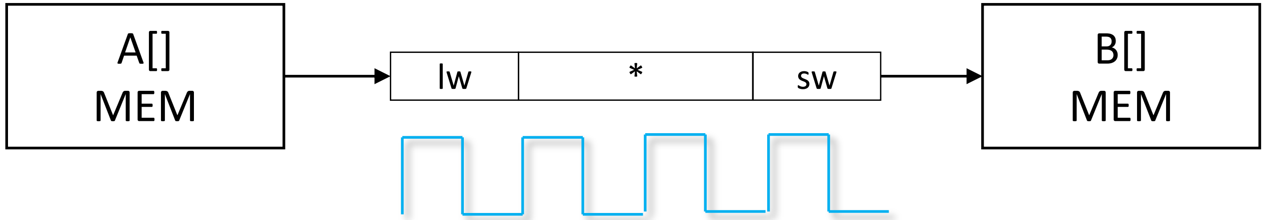
- ✓ **Pros**
  - **Time-to-market**
  - **Quick DSE**
- ✓ **Cons**
  - **Poor QoR**
  - **Fails on large designs**

HLS “Hello World”— *Baseline design*




```
int main () {
...
producer(A,B);
...
}
```

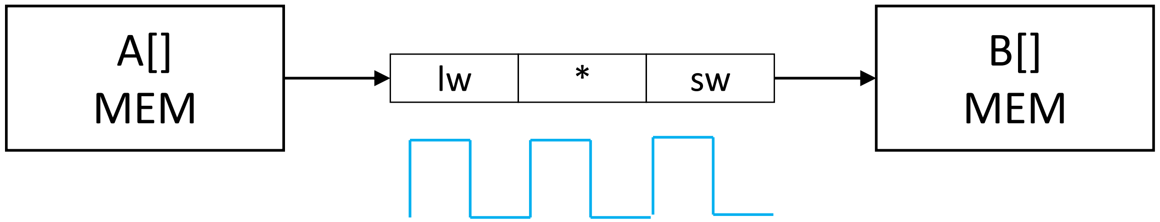
```
void producer(
   int A[8],
   int B[8]) {

   for(int i=0;i<8;i++){
       B[i]=A[i]*3;
   }
}
```



4 cycles*8 iterations=32 clock cycles

HLS “Hello World”— *Bit accurate design*

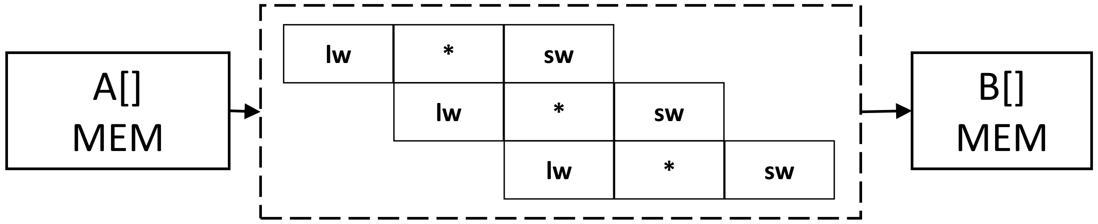
```
void producer(
    ap_int<17> A[8],
    ap_int<17> B[8]) {

    for(int i=0;i<8;i++){
        B[i]=A[i]*3;
    }
}
```



3 cycles*8 iterations=24 clock cycles

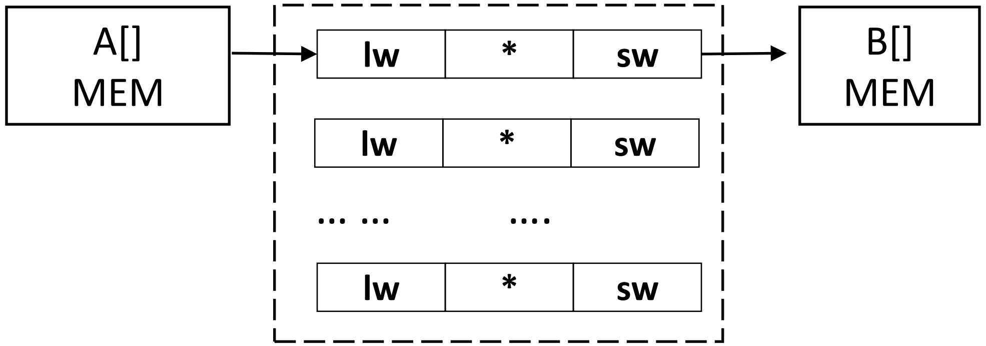
HLS "Hello World"— *Optimized design*

```
void producer(
   ap_int<17> A[8],
   ap_int<17> B[8]) {
   for(int i=0;i<8;i++){
   #pragma pipeline II=1
       B[i]=A[i]*3;
   }
}
```



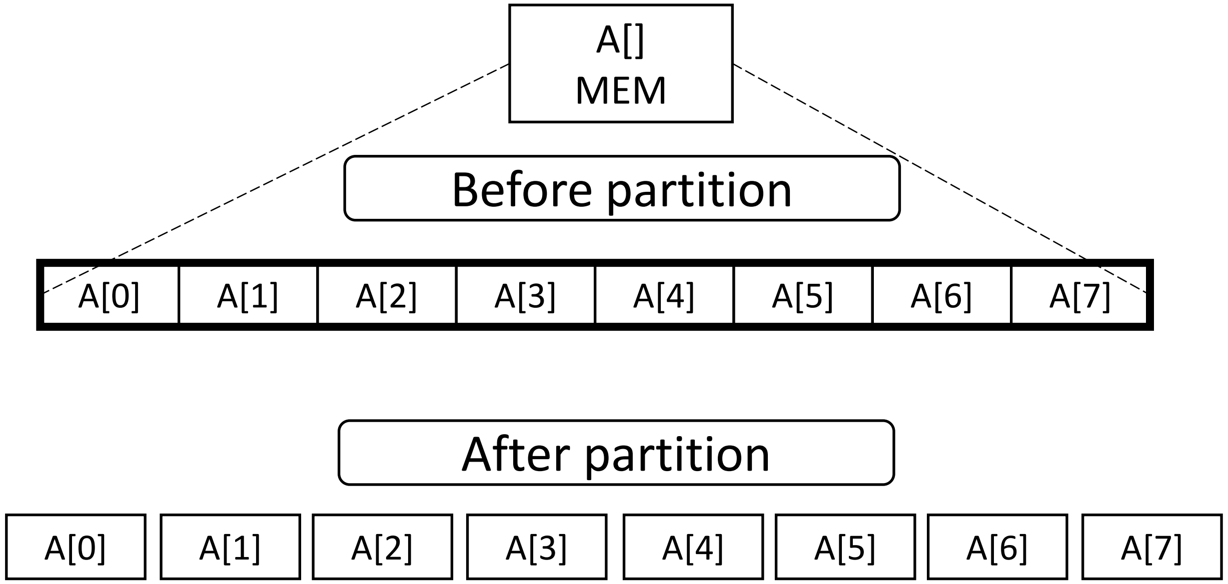
10 clock cycles
1 multiplier

```
void producer(
   ap_int<17> A[8],
   ap_int<17> B[8]) {
   #pragma unroll
    for(int i=0;i<8;i++){
    B[i]=A[i]*3;
   }
}
```

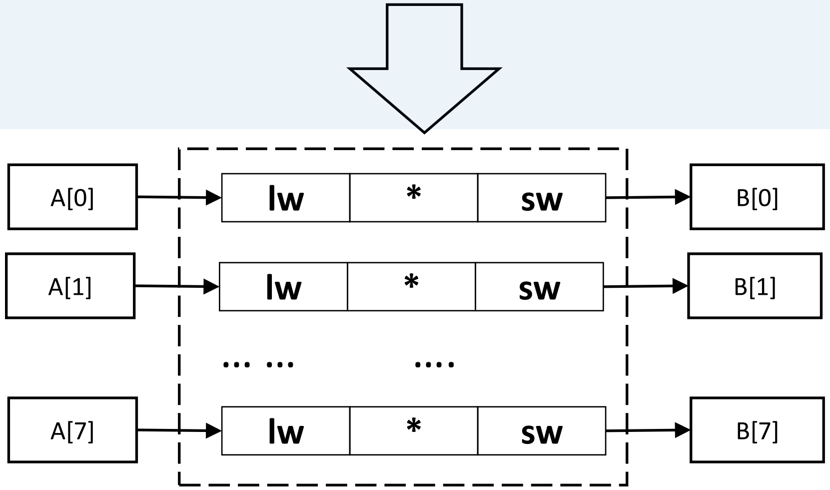


We might, 3 clock cycles
8 multiplier

HLS “Hello World”— *Optimized design*




```
void producer(
   ap_int<17> A[8],
   ap_int<17> B[8]) {
   #pragma unroll
   #pragma  PARTITION variable=B  complete
   #pragma  PARTITION variable=A  complete

    for(int i=0;i<8;i++){
    B[i]=A[i]*3;
   }
}
```



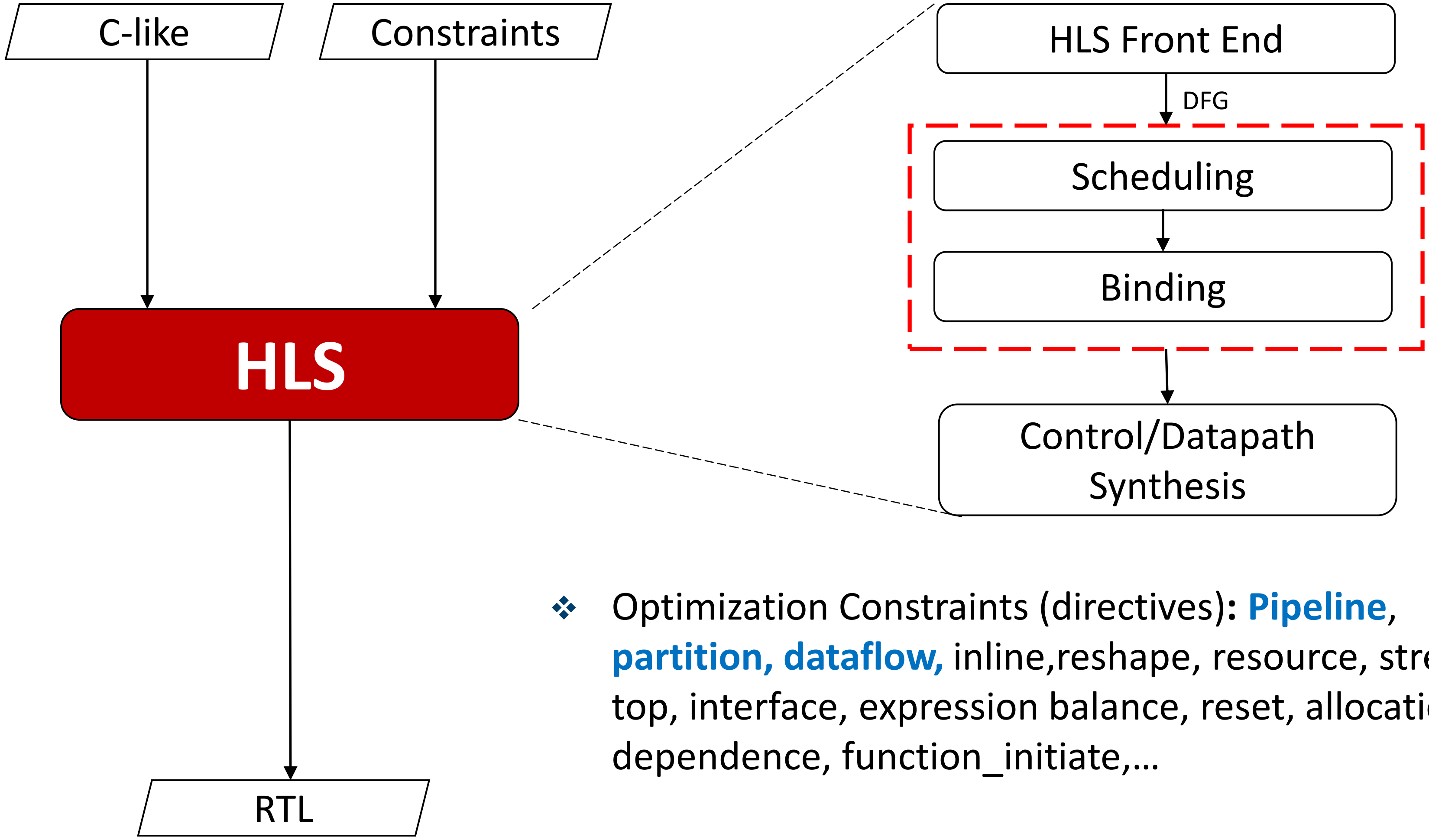
3 clock cycles
8 multiplier

Using HLS for Real World Applications

HLS design flow=

+ Baseline design
+ Bit Accurate design
+ Optimized design (Pipeline,..)

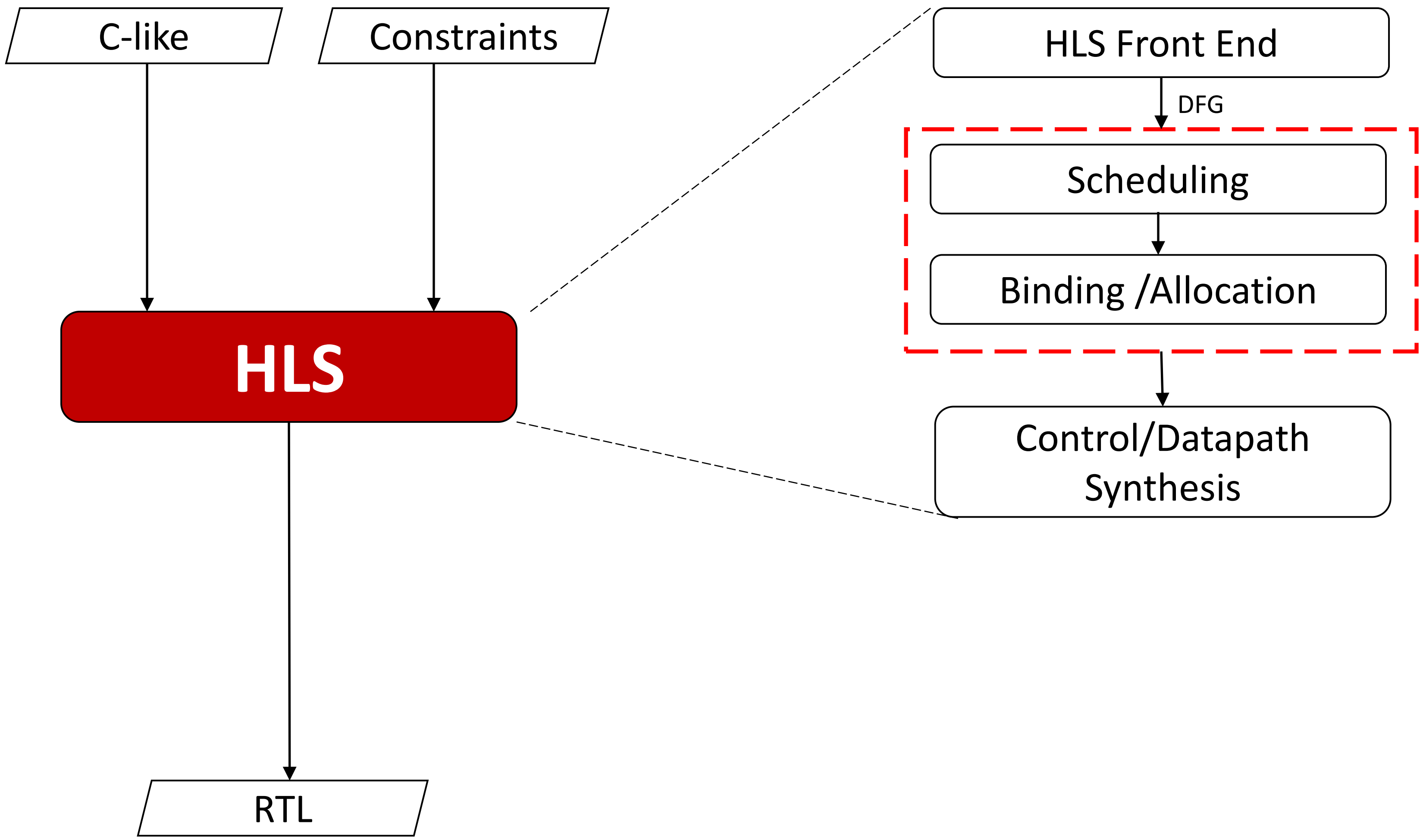
High-Level Synthesis



- Optimization Constraints (directives): **Pipeline**, **partition, dataflow,** inline,reshape, resource, stream, top, interface, expression balance, reset, allocation, dependence, function_initiate,…

High-Level Synthesis Scheduling, Binding, Allocation

High-Level Synthesis



Scheduling in HLS

Scheduling determines **start** and **end** time of “Operations”

Scheduling in HLS

Scheduling determines **start** and **end** time of “Operations”

Goal: High performance and Low area

Scheduling in HLS: Example

A = B *C

D = E* F

J = A* D

T = J- Y

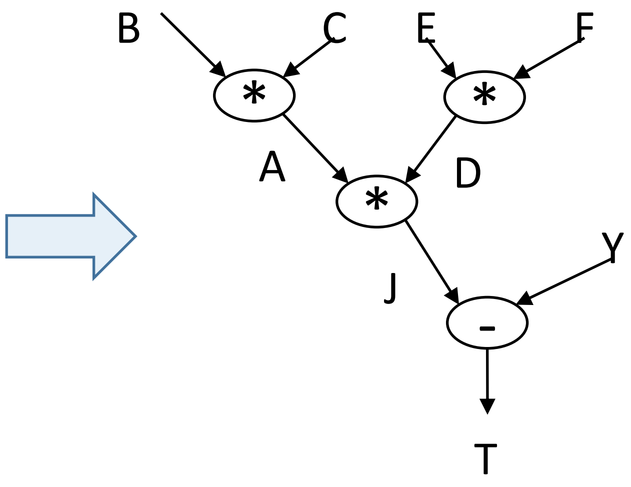
Scheduling in HLS: Example

A = B *C

D = E* F

J = A* D

T = J- Y



Control (Dataflow) Graph

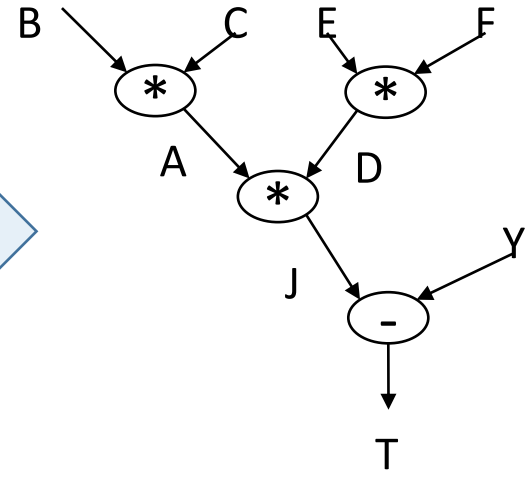
Scheduling in HLS: Example

A = B *C

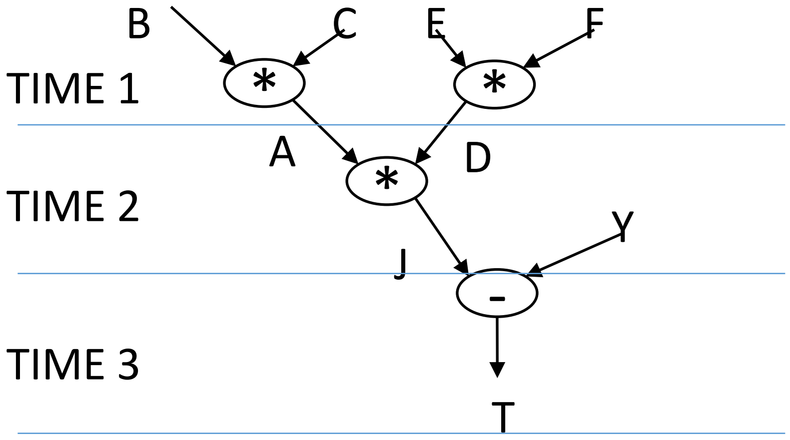
D = E* F

J = A* D

T = J- Y



Control (Dataflow) Graph



2 multipliers
Time: 3

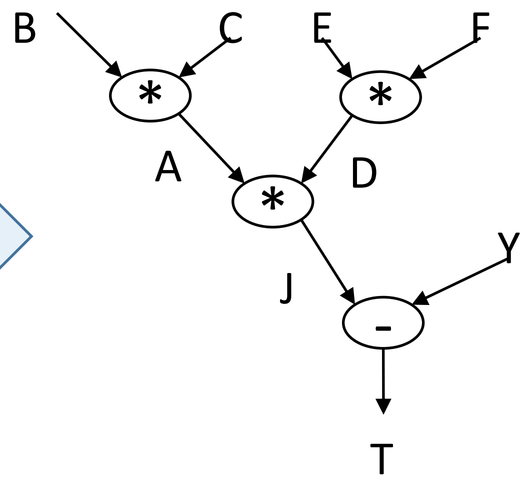
Scheduling in HLS: Example

A = B *C

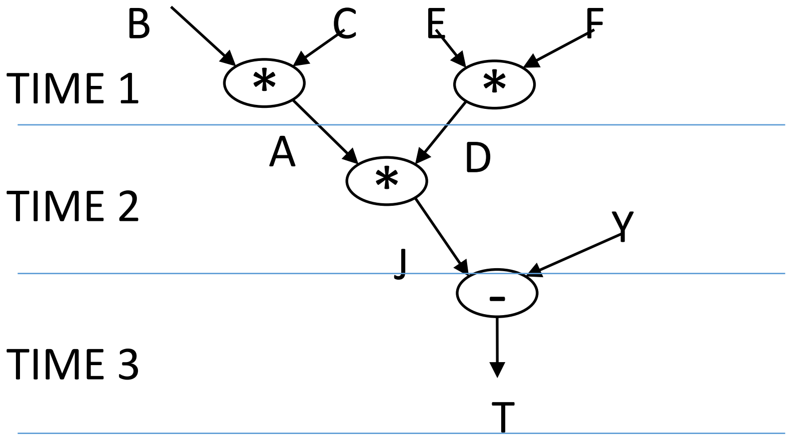
D = E* F

J = A* D

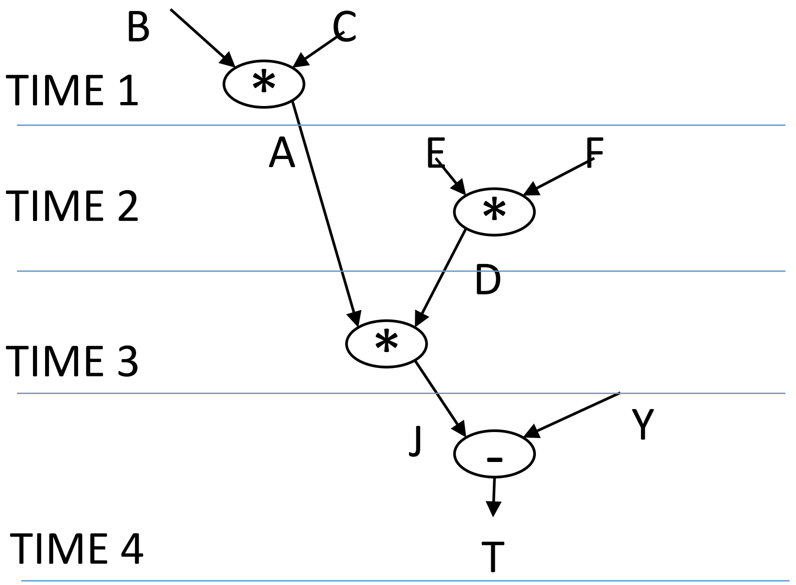
T = J- Y



Control (Dataflow) Graph



2 multipliers
Time: 3



1 multipliers
Time: 4

ASAP Scheduling

Algorithm 1: ASAP Scheduling

Procedure `ASAP()`

Data: $G_S(V, E)$

`/* V: Vertices, E: Edges`

Result: t

Schedule v_0 by setting $t_0 = 1$

repeat

Select a v_i whose predecessors are all scheduled;

Schedule v_i by setting $t_i = max(t_j) + d_j$

where $j : (v_j, v_i) \in E$

until v_n *is scheduled*;

ASAP Scheduling

Algorithm 1: ASAP Scheduling

Procedure `ASAP()`

  Data: $G_S(V,E)$

  `/* V: Vertices, E: Edges`

  Result: t

  Schedule v_0 by setting $t_0 = 1$

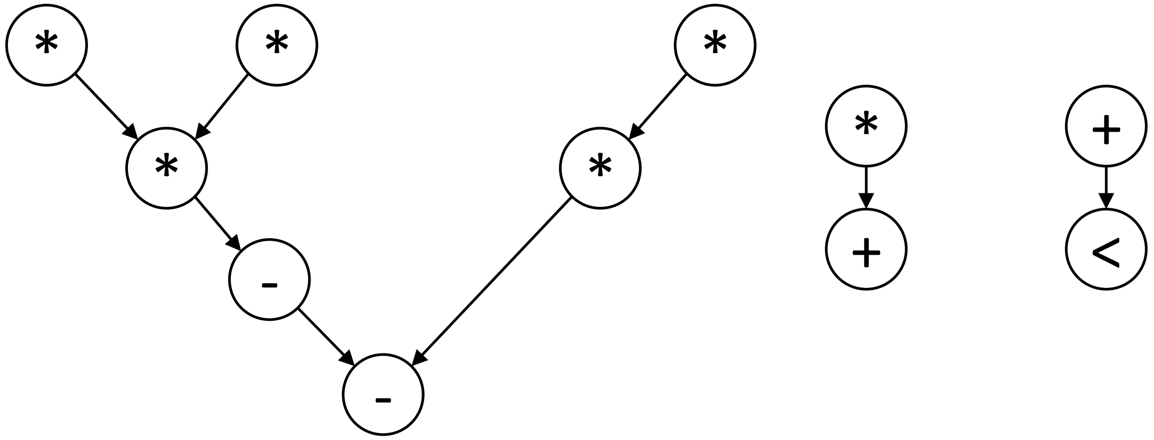
  repeat

    Select a v_i whose predecessors are all scheduled;

    Schedule v_i by setting $t_i = max(t_j) + d_j$

    where $j : (v_j, v_i) \in E$

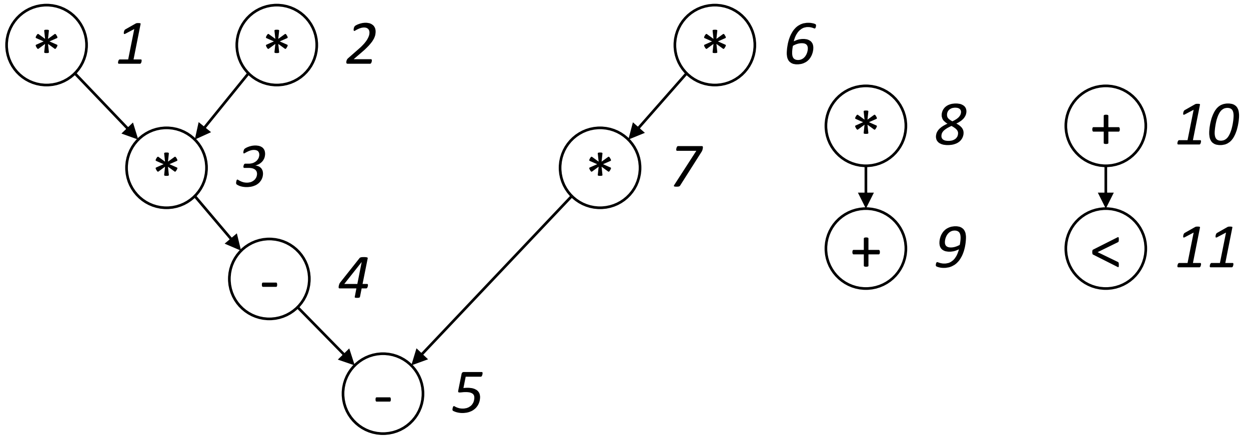
  until v_n *is scheduled*;



ASAP Scheduling

Algorithm 1: ASAP Scheduling

Procedure `ASAP()`
    Data: $G_S(V,E)$
    `/* V: Vertices, E: Edges`
    Result: t
    Schedule v_0 by setting $t_0 = 1$
    repeat
        Select a v_i whose predecessors are all scheduled;
        Schedule v_i by setting $t_i = max(t_j) + d_j$
        where $j : (v_j, v_i) \in E$
    until v_n *is scheduled*;



ASAP Scheduling

Algorithm 1: ASAP Scheduling

Procedure `ASAP()`

  Data: $G_S(V,E)$

  `/* V: Vertices, E: Edges`

  Result: t

  Schedule v_0 by setting $t_0 = 1$

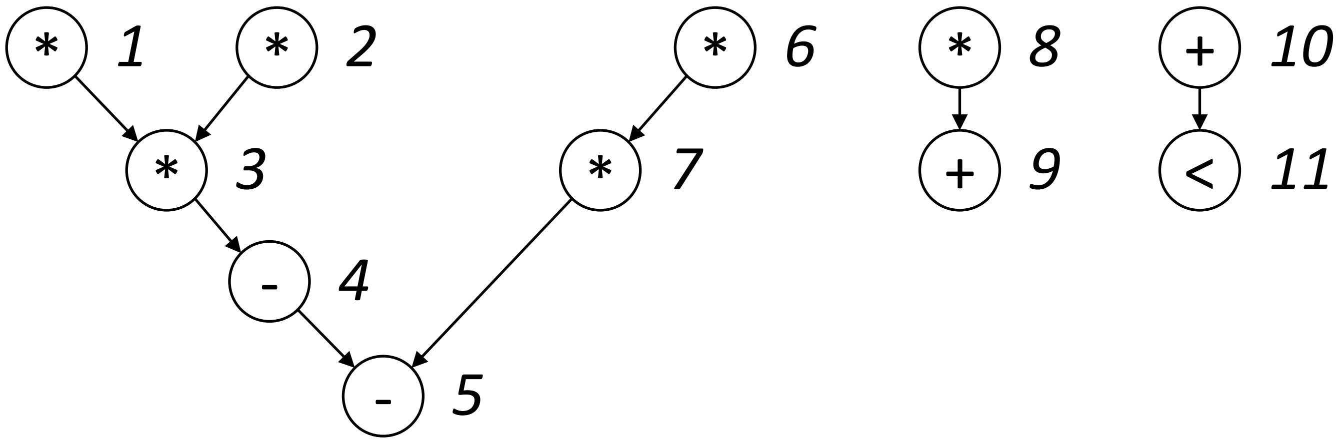
  repeat

    Select a v_i whose predecessors are all scheduled;

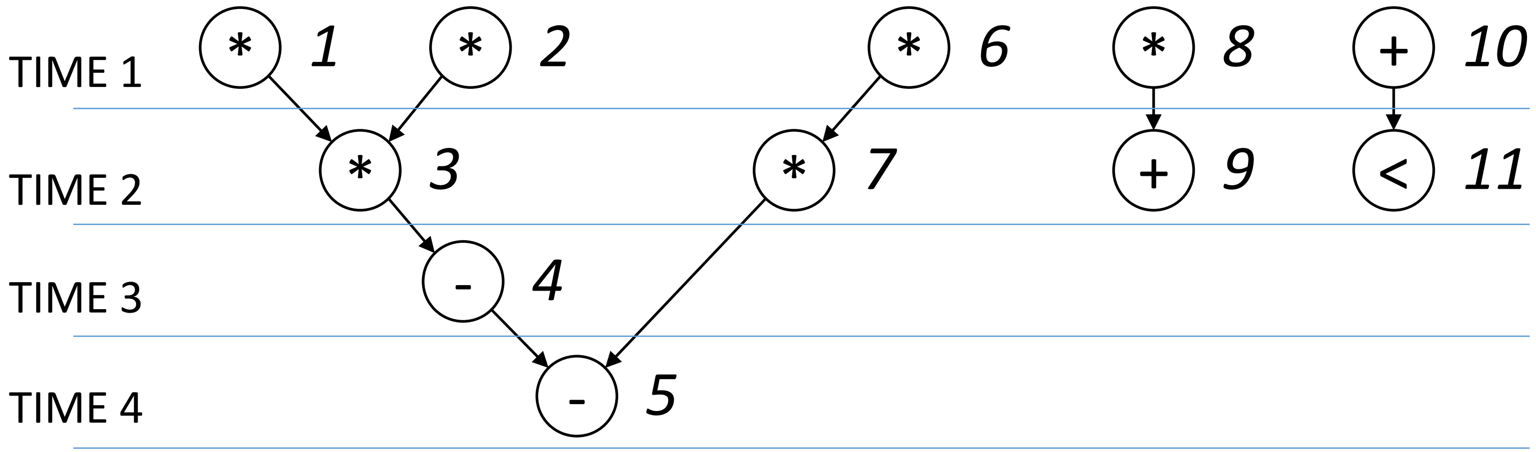
    Schedule v_i by setting $t_i = max(t_j) + d_j$

    where $j : (v_j, v_i) \in E$

  until v_n *is scheduled*;



ASAP Scheduling



ALAP Scheduling

Algorithm 2: ALAP Scheduling

Procedure `ALAP()`

  Data: $G_S(V, E)$

  `/* V: Vertices, E: Edges`

  Result: t

  Schedule v_n by setting $t_n = \lambda + 1$

  repeat

    Select a v_i successors are all scheduled;

    Schedule v_i by setting $t_i = min(t_j) - d_i$

    where $j : (v_i, v_j) \in E$

  until v_0 *is scheduled*;

ALAP Scheduling

Algorithm 2: ALAP Scheduling

Procedure `ALAP()`

  Data: $G_S(V,E)$

  `/* V: Vertices, E: Edges`

  Result: t

  Schedule v_n by setting $t_n = \lambda + 1$

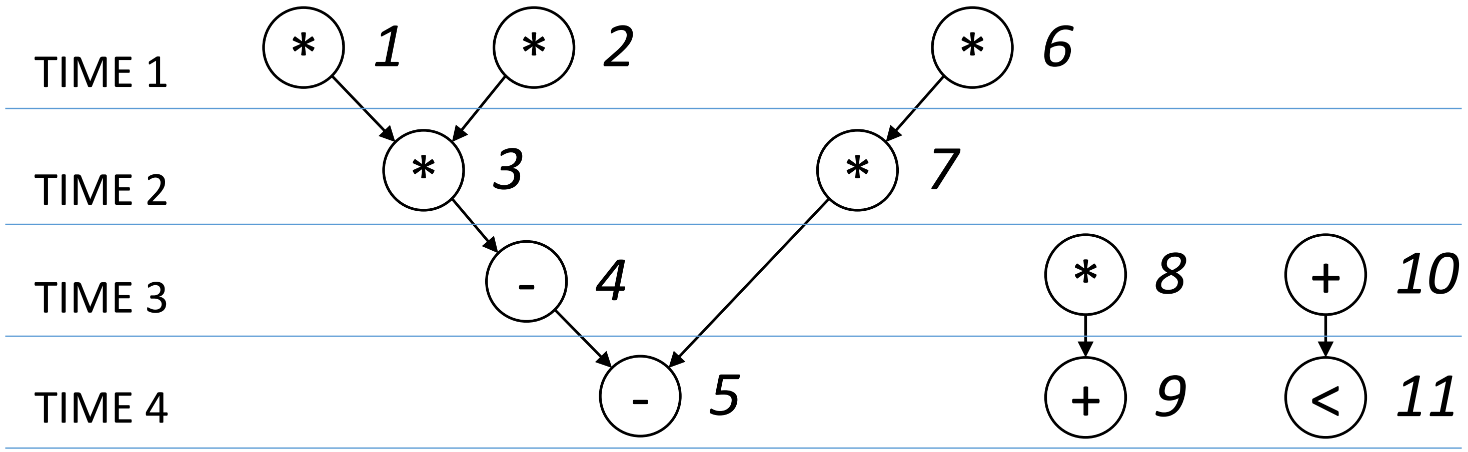
  repeat

    Select a v_i successors are all scheduled;

    Schedule v_i by setting $t_i = min(t_j) - d_i$

    where $j : (v_i, v_j) \in E$

  until v_0 *is scheduled*;



Force-Directed Scheduling

Algorithm 3: Force Directed Scheduling

Procedure `ForceDirectedScheduling()`

  Data: $G_S(V,E)$

  `/* V: Vertices, E: Edges`

  Result: t

  repeat

    Compute time frames

    Compute operation probabilities

    Compute DG

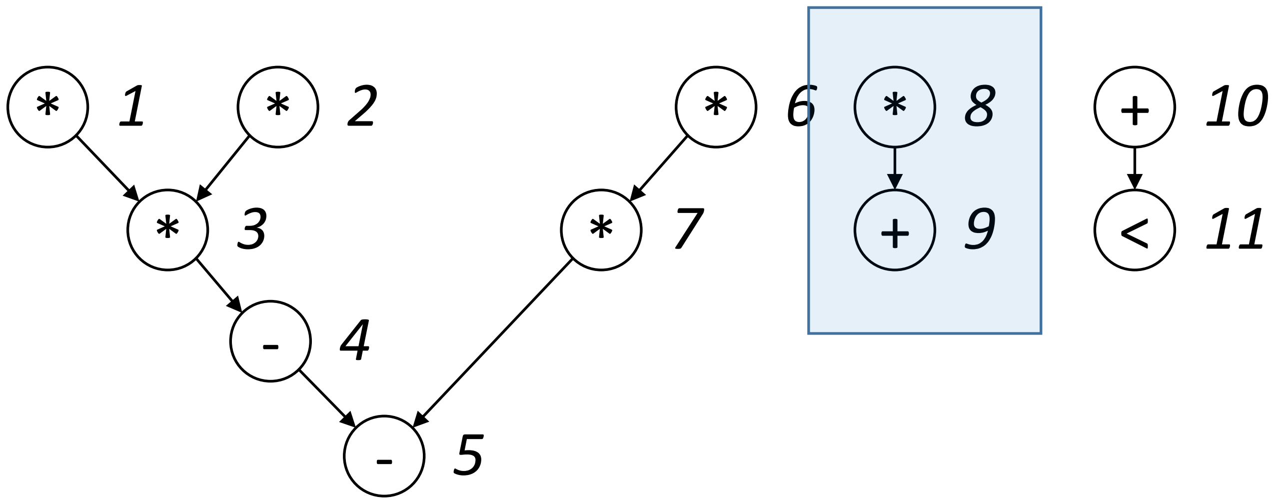
    Compute self-force, succ/pred force

    Schedule the operation with least force

  until *all operations are scheduled*;

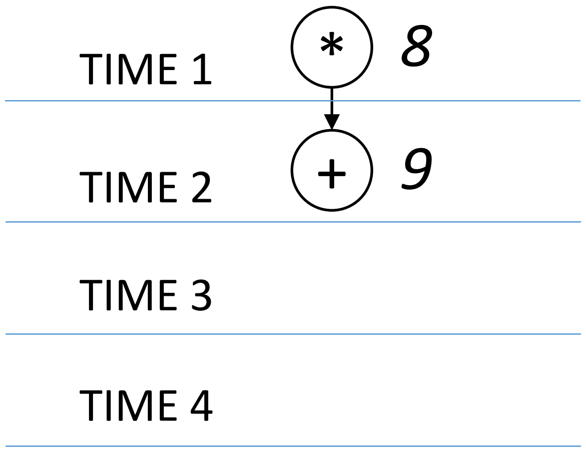
Time frame calculation

Time frame = {ASAP, ALAP }

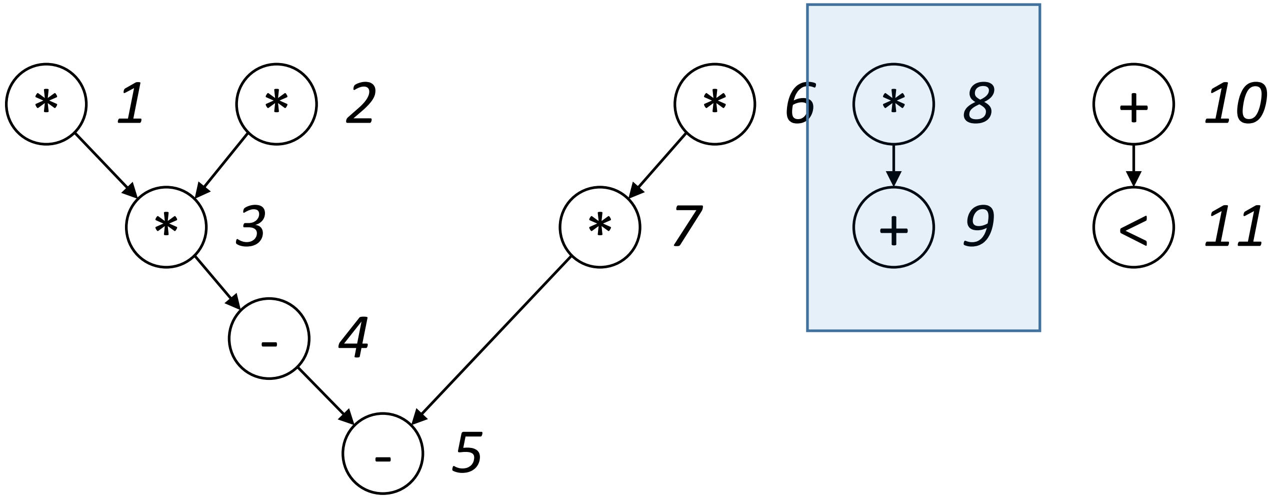


Time frame calculation

Time frame = {ASAP, ALAP }

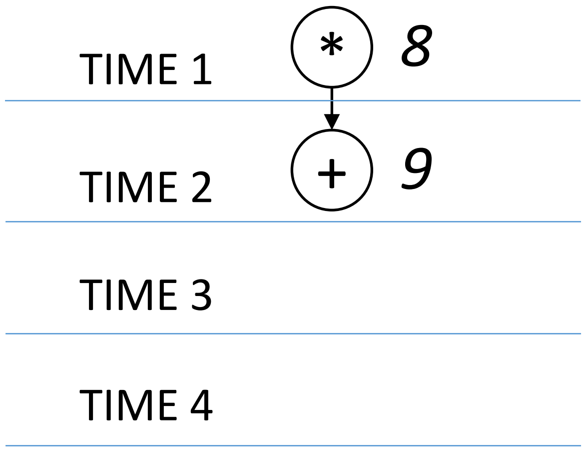


ASAP

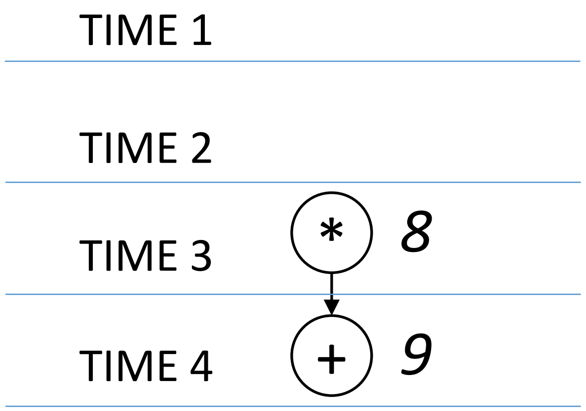


Time frame calculation

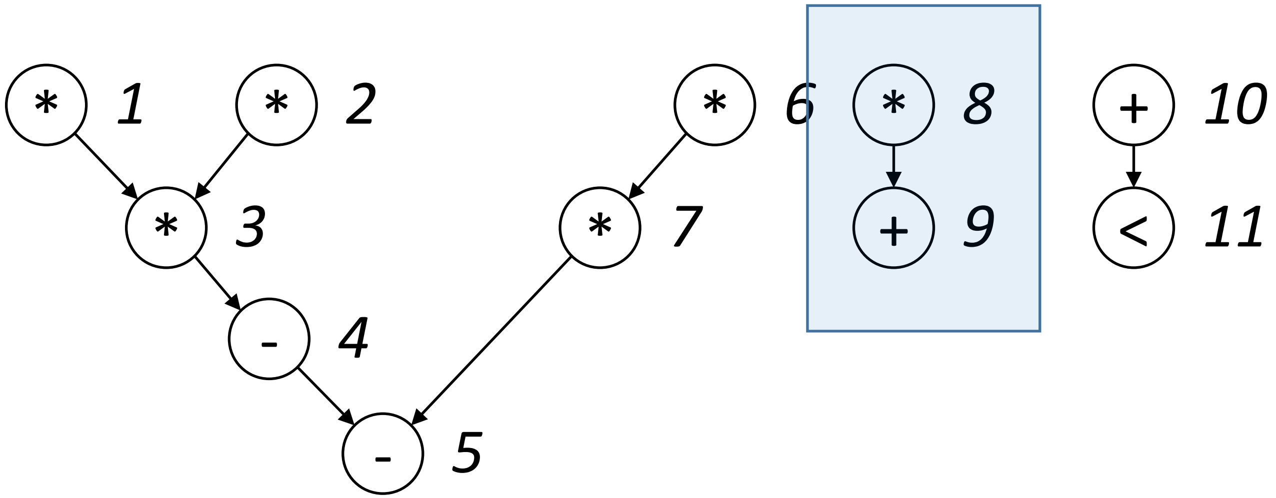
Time frame = {ASAP, ALAP }



ASAP

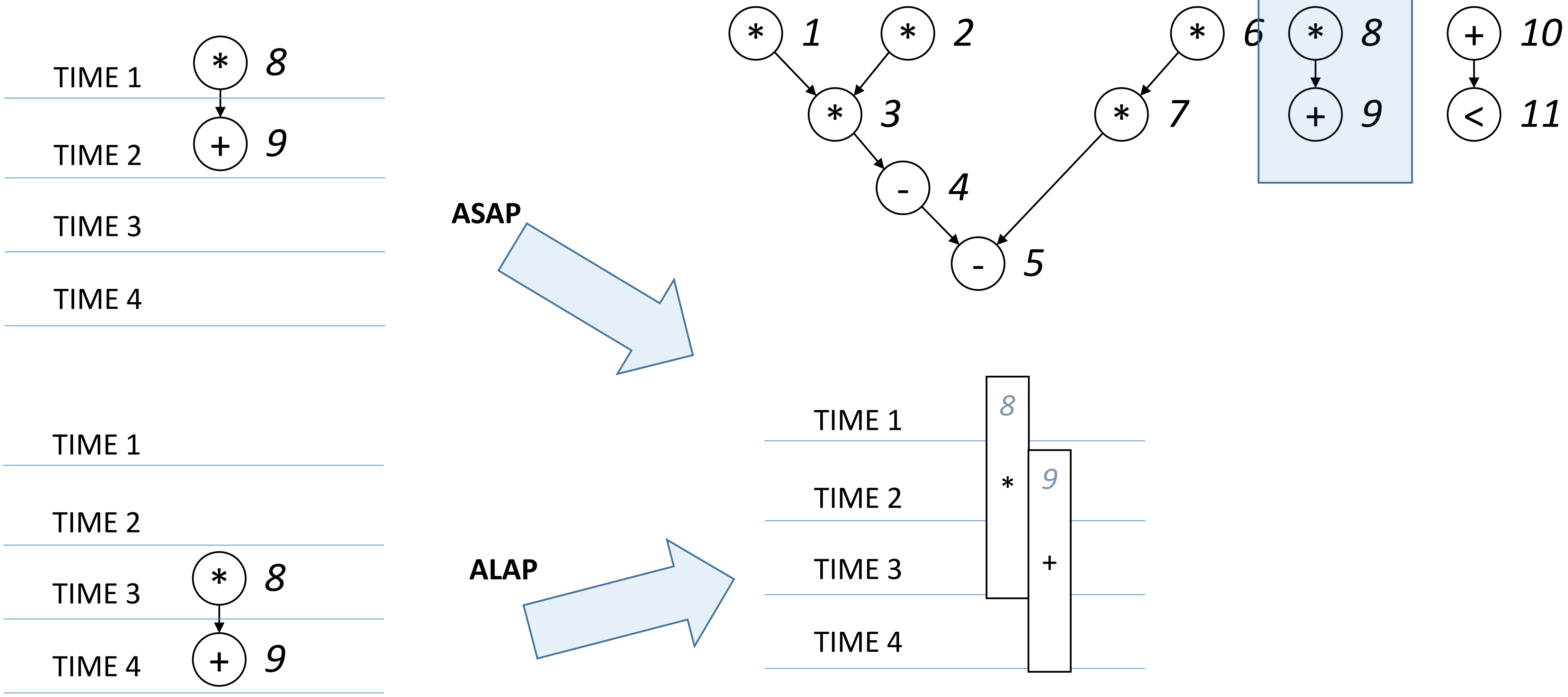


ALAP

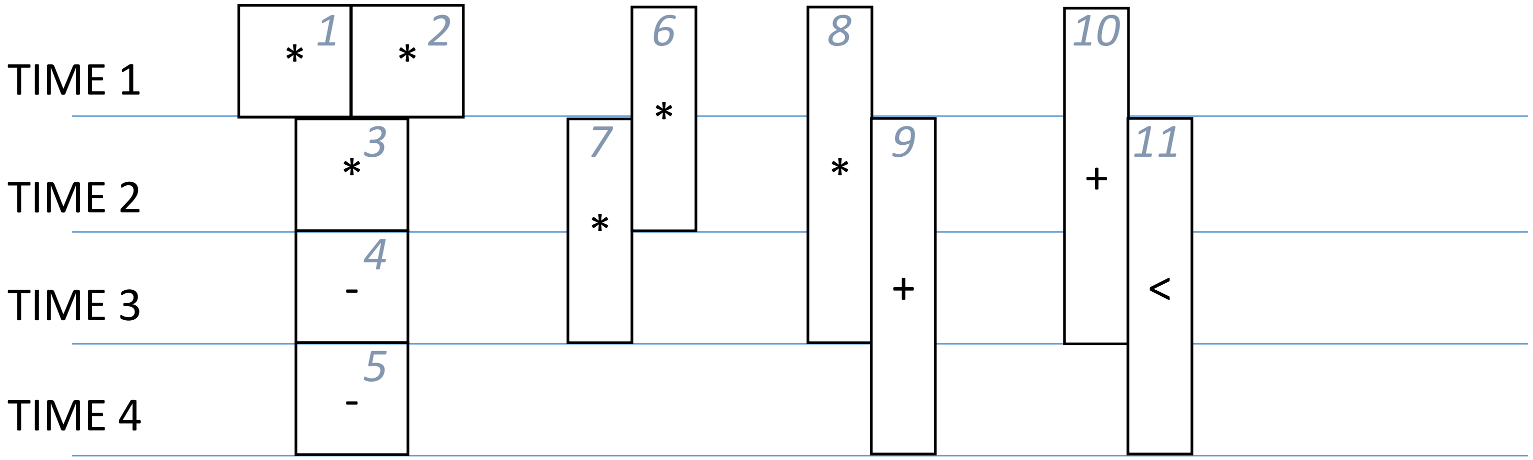


Time frame calculation

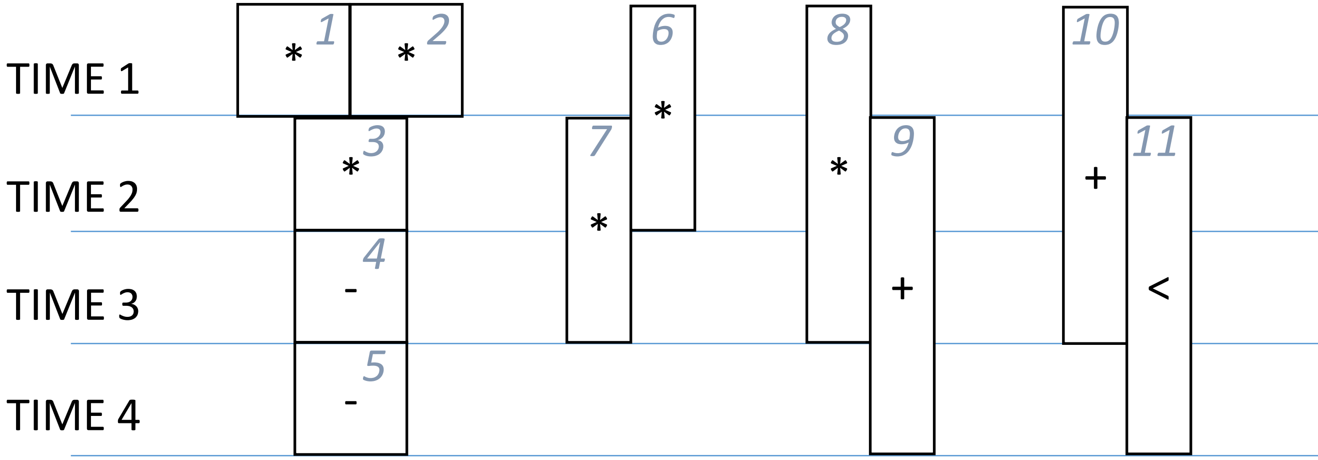
Time frame = {ASAP, ALAP }



Time frame calculation

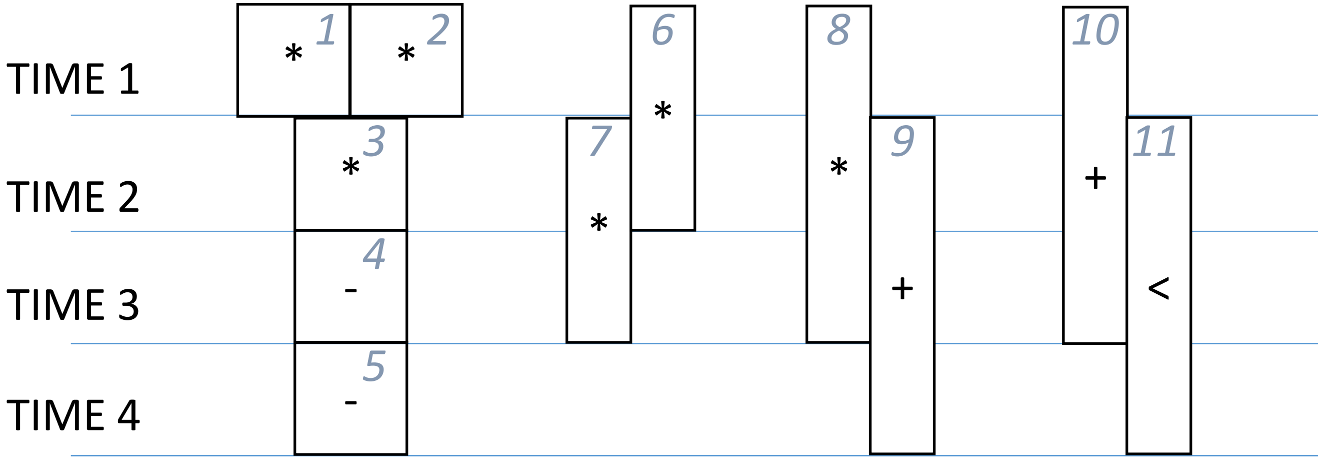


Operation Probability



Prob(i, L) to denote the **operation probability** of an operation *i* at time L.

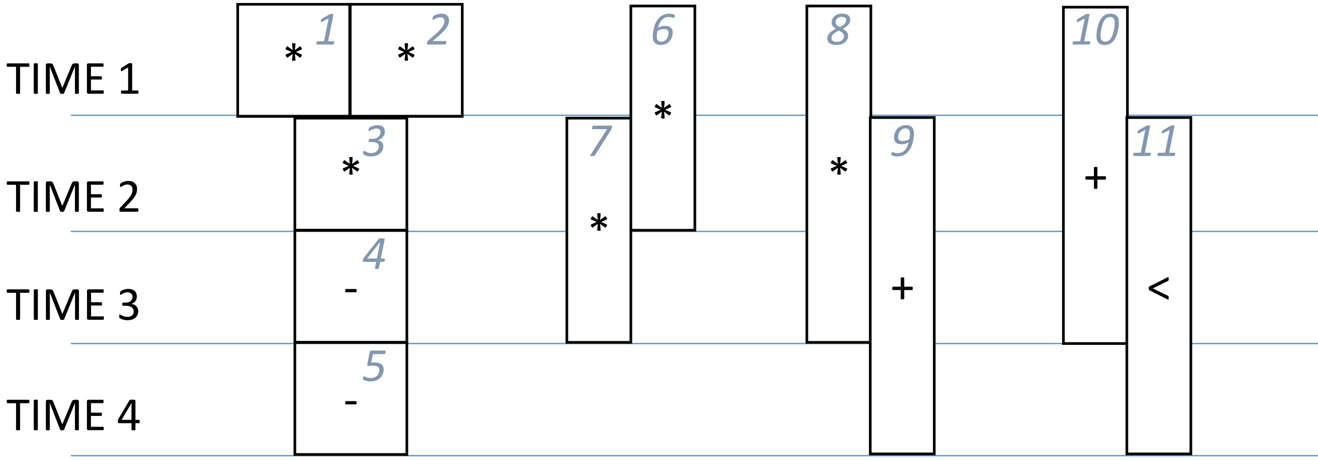
Operation Probability



Prob(1, 1) = 1.0

Prob(i, L) to denote the **operation probability** of an operation *i* at time L.

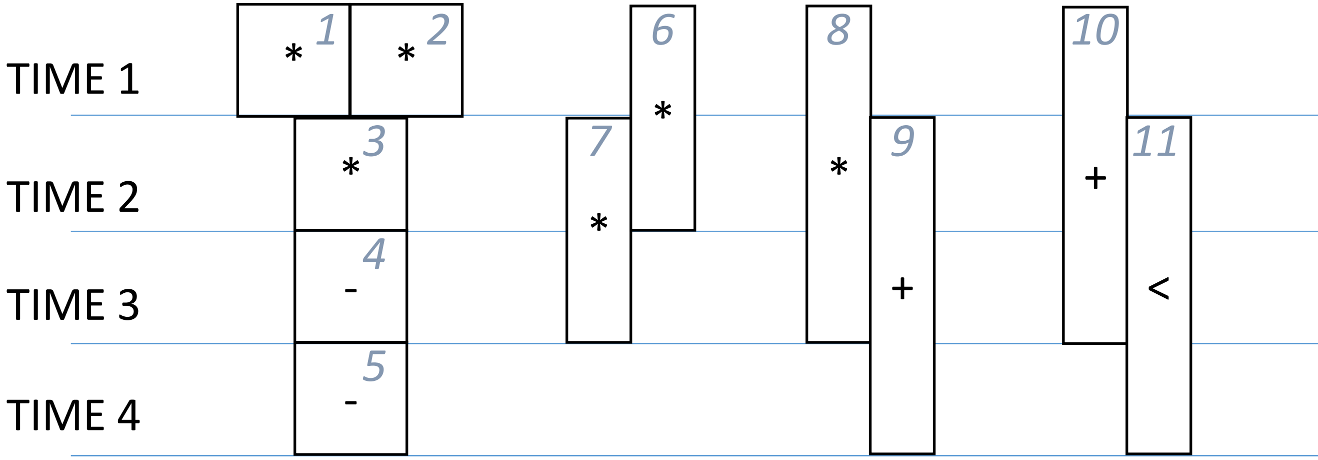
Operation Probability



Prob(1, 1) = 1.0
Prob(1, 2) = 0.0
Prob(1, 3) = 0.0
Prob(1, 4) = 0.0

Prob(i, L) to denote the **operation probability** of an operation *i* at time L.

Operation Probability

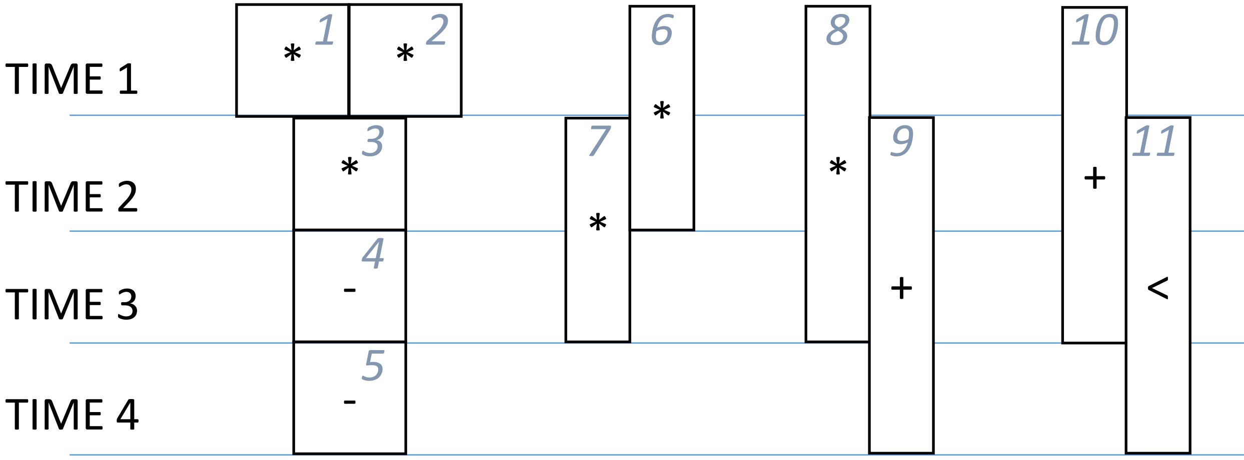


Prob(1, 1) = 1.0	Prob(6, 1) = 0.5	Prob(8, 1) = 1/3
Prob(1, 2) = 0.0	Prob(6, 2) = 0.5	Prob(8, 2) = 1/3
Prob(1, 3) = 0.0	Prob(6, 3) = 0.0	Prob(8, 3) = 1/3
Prob(1, 4) = 0.0	Prob(6, 4) = 0.0	Prob(8, 4) = 0.0

Prob(i, L) to denote the **operation probability** of an operation *i* at time L.

Type Distribution Graph

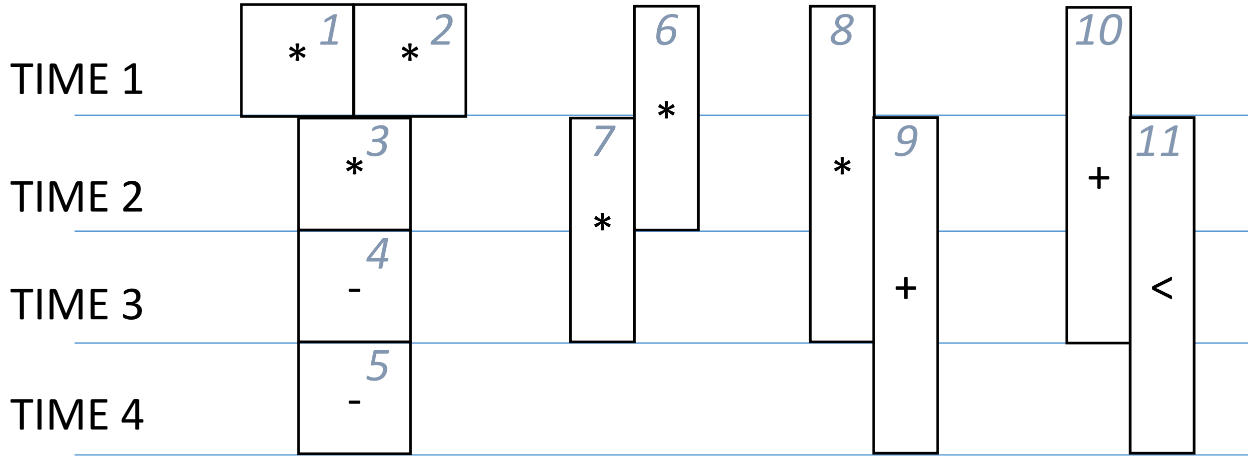
❖The type distribution is the sum of probabilities of an operation that can be executed with a specific hardware resource at time L.



Type Distribution Graph

❖The type distribution is the sum of probabilities of an operation that can be executed with a specific hardware resource at time L.

✓DG(k, L) : distribution of resource k at time L.

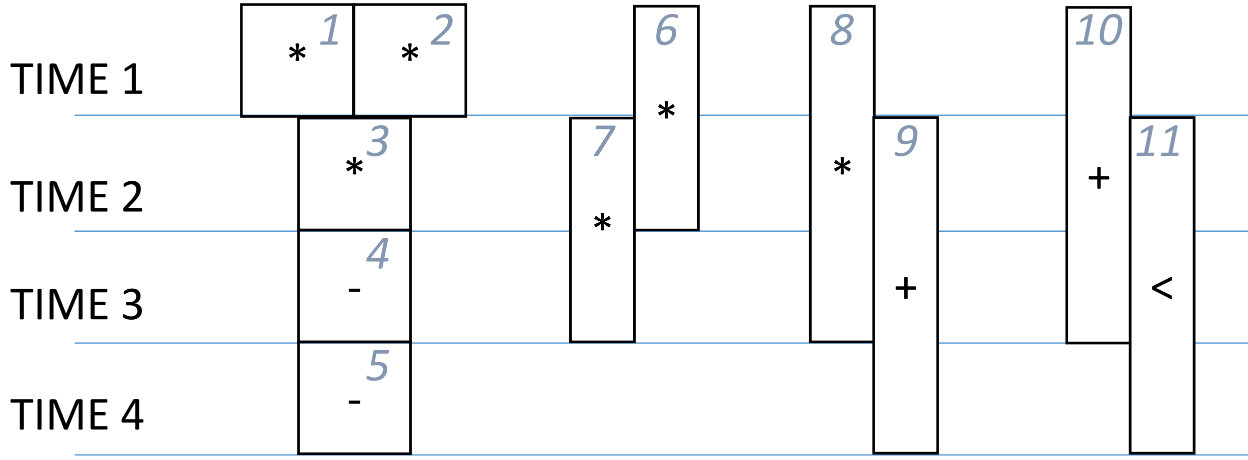


Type Distribution Graph

❖The type distribution is the sum of probabilities of an operation that can be executed with a specific hardware resource at time L.

✓DG(k, L) : distribution of resource k at time L.

✓Resources: multipliers, address, registers,..

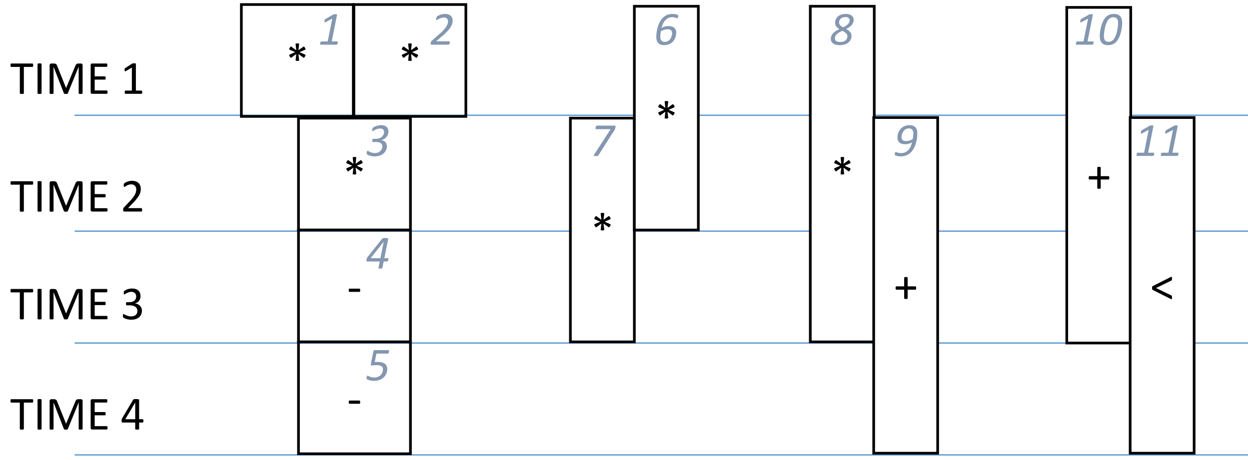


Type Distribution Graph

❖The type distribution is the sum of probabilities of an operation that can be executed with a specific hardware resource at time L.

✓DG(k, L) : distribution of resource k at time L.

✓Resources: multipliers, address, registers,..



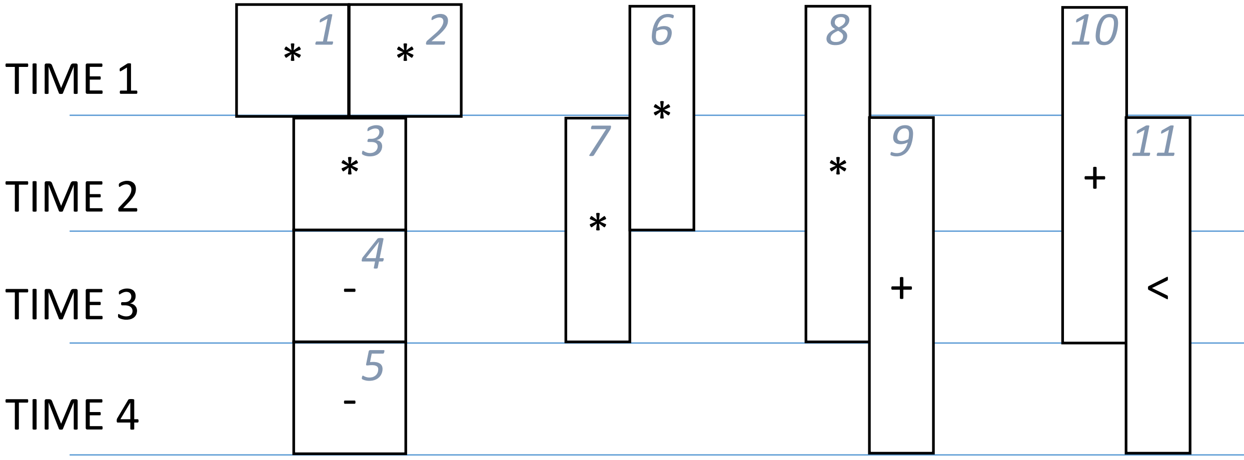
$$DG(k, L) \; = Sum\ (Prob\ (i, L))\ for\ all\ i$$

Type Distribution Graph

❖The type distribution is the sum of probabilities of an operation that can be executed with a specific hardware resource at time L.

✓DG(k, L) : distribution of resource k at time L.

✓Resources: multipliers, address, registers,..



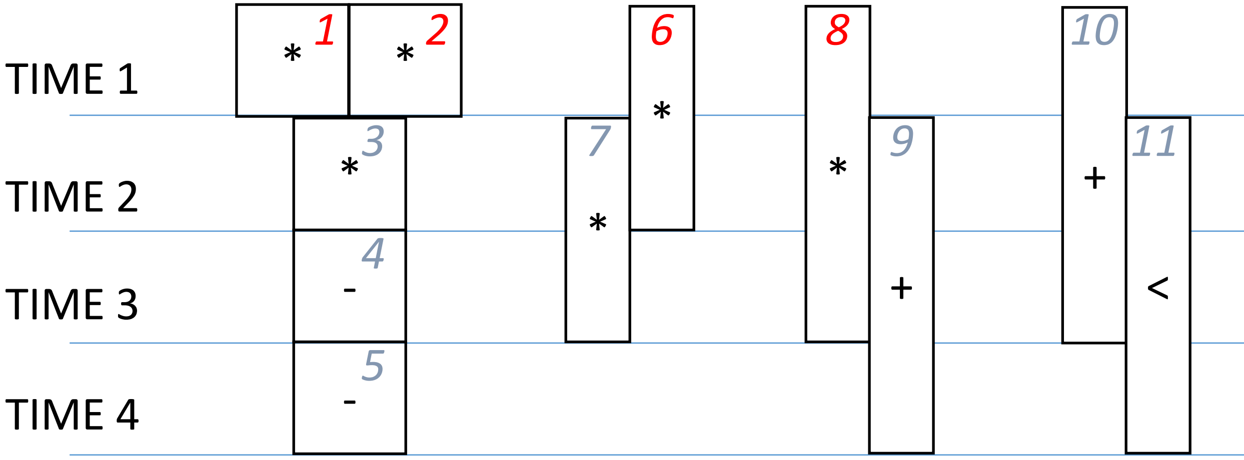
$$DG(k{=}multiplier,\ 1)\ =\ ?$$

Type Distribution Graph

❖The type distribution is the sum of probabilities of an operation that can be executed with a specific hardware resource at time L.

✓DG(k, L) : distribution of resource k at time L.

✓Resources: multipliers, address, registers,..



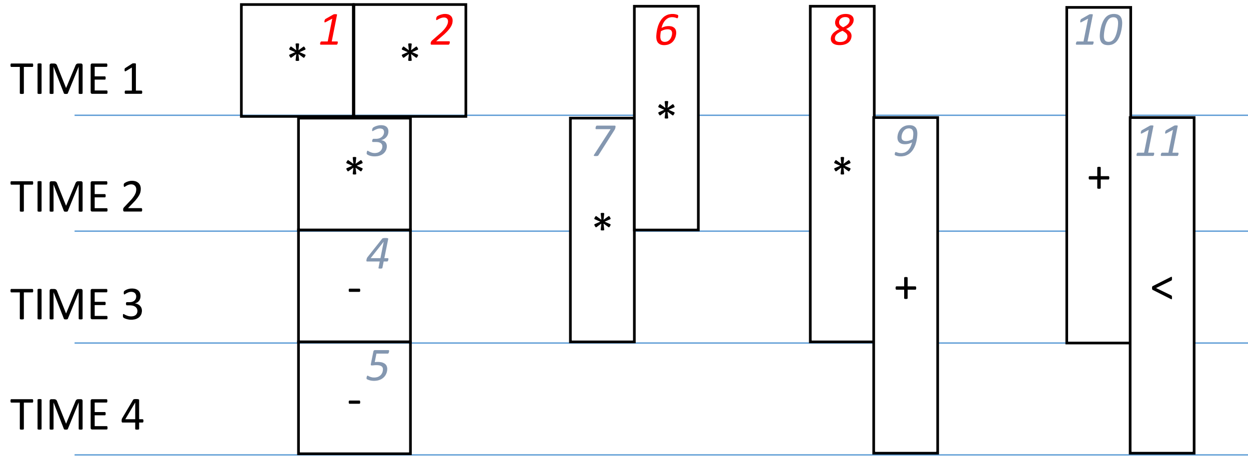
E.g., TIME 1 → Operations 1, 2, 6 and 8 can be scheduled on a multiplier.

Type Distribution Graph

❖The type distribution is the sum of probabilities of an operation that can be executed with a specific hardware resource at time L.

✓DG(k, L) : distribution of resource k at time L.

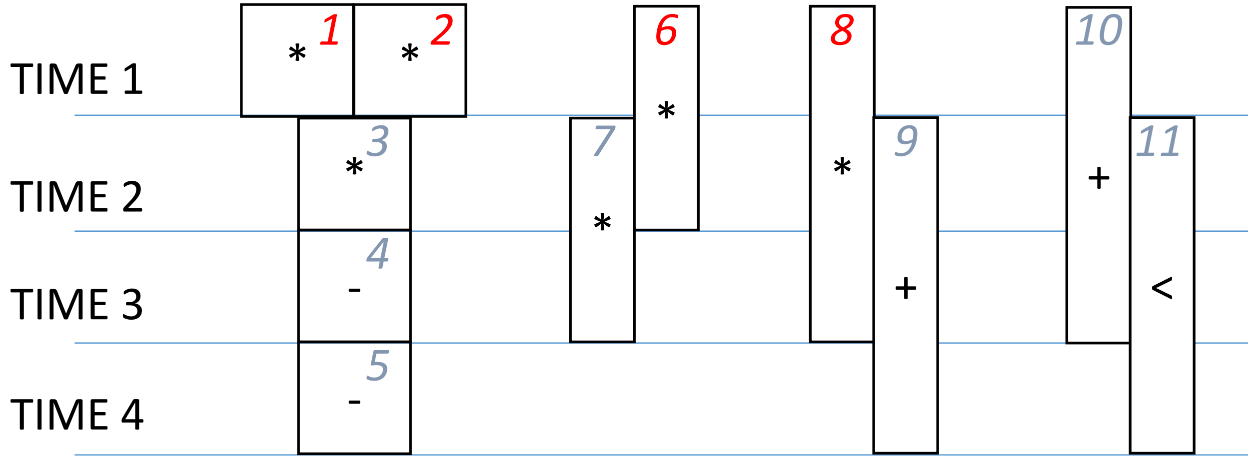
✓Resources: multipliers, address, registers,..



E.g., TIME 1 → Operations 1, 2, 6 and 8 can be scheduled on a multiplier.
DG(k=multiplier, 1) = Prob (1,1) + Prob(1,2) + Prob (6, 1) + Prob (8,1)

Type Distribution Graph

- ❖The type distribution is the sum of probabilities of an operation that can be executed with a specific hardware resource at time L.
  - ✓DG(k, L) : distribution of resource k at time L.
  - ✓Resources: multipliers, address, registers,..



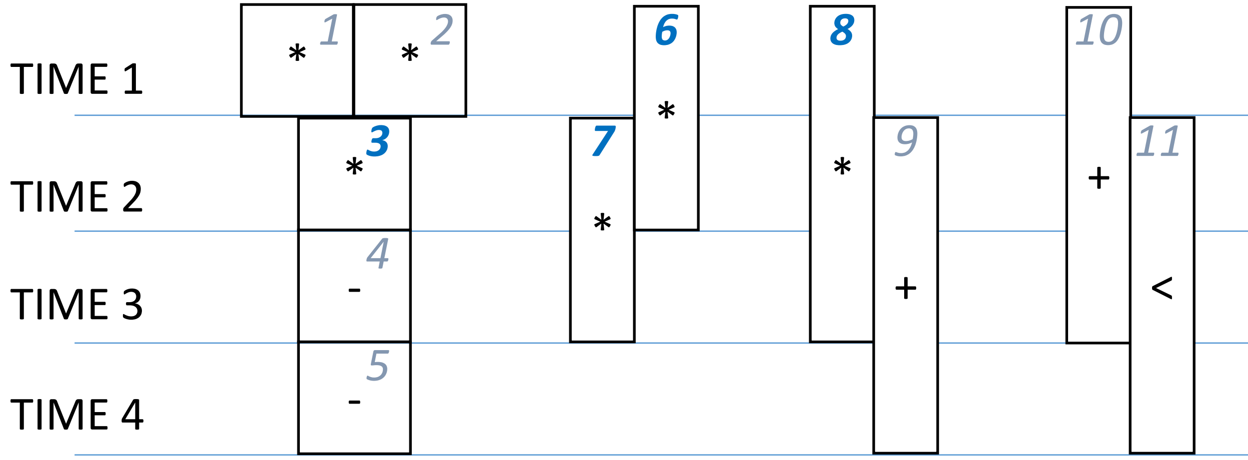
E.g., TIME 1 → Operations 1, 2, 6 and 8 can be scheduled on a multiplier.
DG(k=multiplier, 1) = Prob (1,1) + Prob(1,2) + Prob (6, 1) + Prob (8,1) = 1.0 + 1.0 + 0.5 +0.3 = 2.8

Type Distribution Graph

❖The type distribution is the sum of probabilities of an operation that can be executed with a specific hardware resource at time L.

✓DG(k, L) : distribution of resource k at time L.

✓Resources: multipliers, address, registers,..



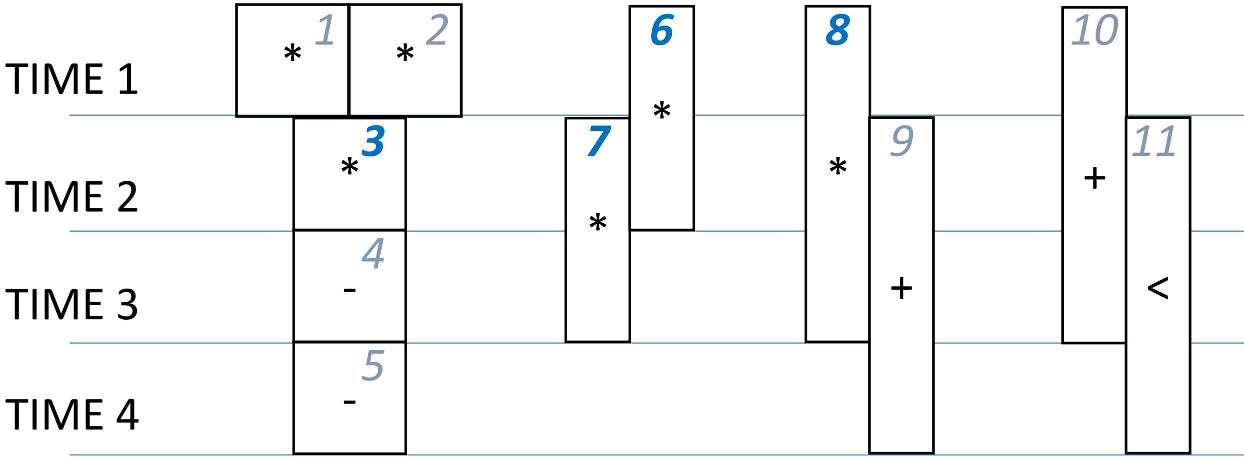
E.g., TIME 1 → Operations 1, 2, 6 and 8 can be scheduled on a multiplier.

DG(k=multiplier, 1) = Prob (1,1) + Prob(1,2) + Prob (6, 1) + Prob (8,1) = 1.0 + 1.0 + 0.5 +0.3 = 2.8

E.g., TIME 2 → Operations **3, 6, 7,** and **8** can be scheduled on a multiplier.

Type Distribution Graph

- ❖The type distribution is the sum of probabilities of an operation that can be executed with a specific hardware resource at time L.
  - ✓DG(k, L) : distribution of resource k at time L.
  - ✓Resources: multipliers, address, registers,..



E.g., TIME 1 → Operations 1, 2, 6 and 8 can be scheduled on a multiplier.
DG(k=multiplier, 1) = Prob (1,1) + Prob(1,2) + Prob (6, 1) + Prob (8,1) = 1.0 + 1.0 + 0.5 +0.3 = 2.8

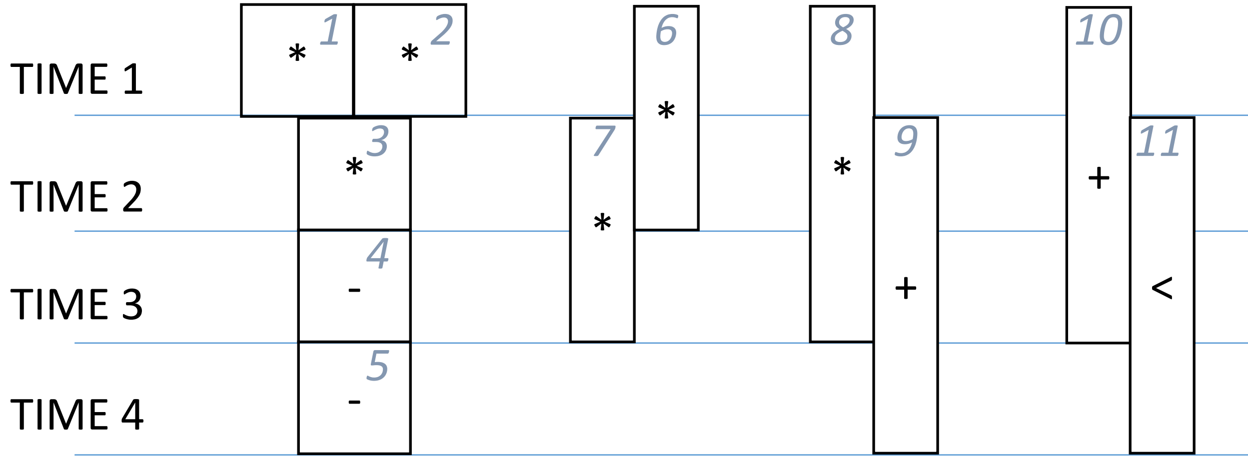
E.g., TIME 2 → Operations **3, 6, 7,** and **8** can be scheduled on a multiplier.
DG(k=multiplier, 2) = Prob (3,2) + Prob(6,2) + Prob (7, 2) + Prob (8,2) = 1.0 + 0.5 + 0.5 +0.3 = 2.3

Type Distribution Graph

❖The type distribution is the sum of probabilities of an operation that can be executed with a specific hardware resource at time L.

✓DG(k, L) : distribution of resource k at time L.

✓Resources: multipliers, address, registers,..



E.g., TIME 1 → Operations 1, 2, 6 and 8 can be scheduled on a multiplier.
DG(k=multiplier, 1) = Prob (1,1) + Prob(1,2) + Prob (6, 1) + Prob (8,1) = 1.0 + 1.0 + 0.5 +0.3 = 2.8

E.g., TIME 2 → Operations 3, 6, 7, and 8 can be scheduled on a multiplier.
DG(k=multiplier, 2) = Prob (3,2) + Prob(6,2) + Prob (7, 2) + Prob (8,2) = 1.0 + 0.5 + 0.5 +0.3 = 2.3

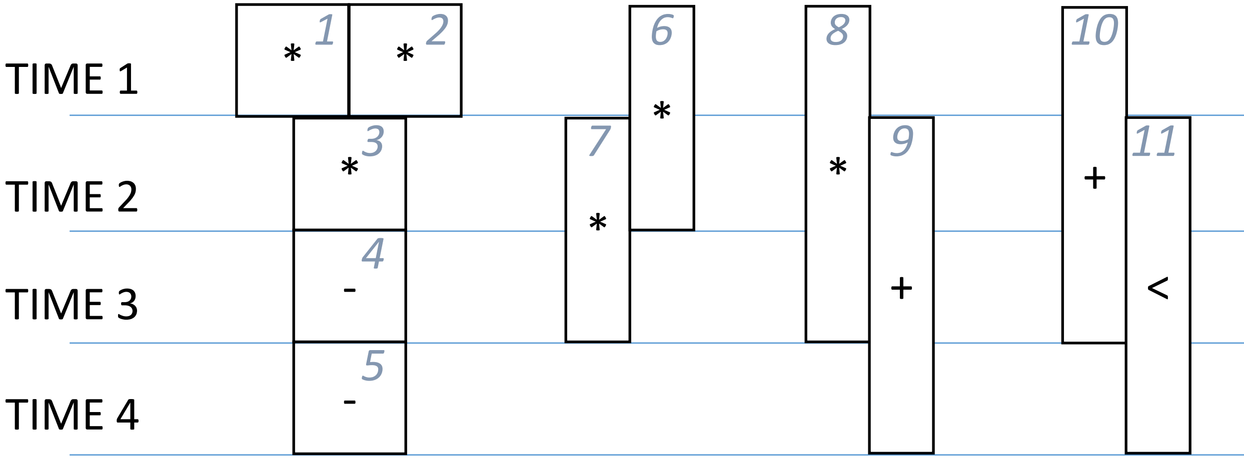
E.g., TIME 3 → Operations 7 and 8 can be scheduled on a multiplier.
DG(k=multiplier, 3) = Prob (7, 3) + Prob (8,3) = 0.5 +0.3 = 0.8

Type Distribution Graph

❖The type distribution is the sum of probabilities of an operation that can be executed with a specific hardware resource at time L.

✓DG(k, L) : distribution of resource k at time L.

✓Resources: multipliers, address, registers,..



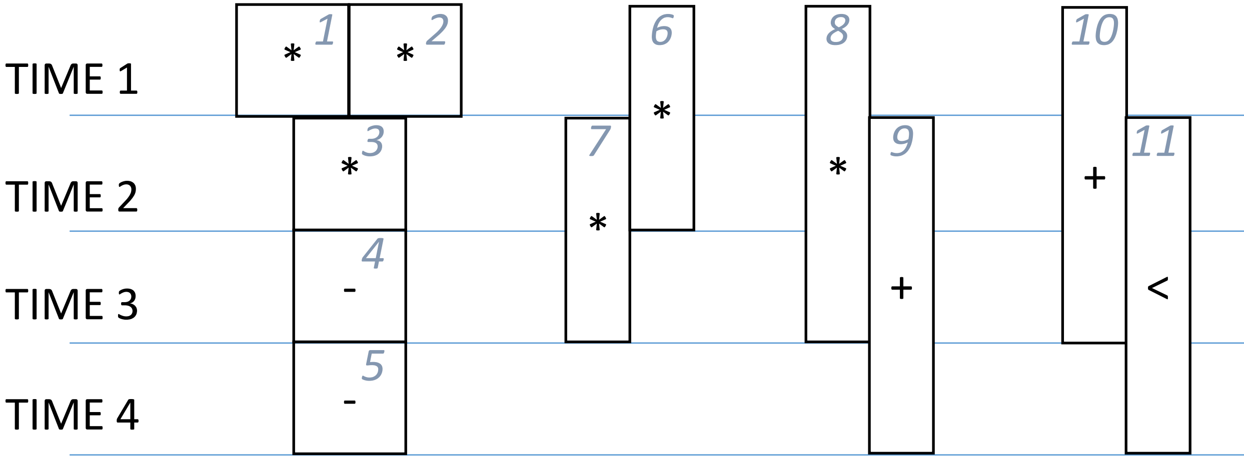
E.g., TIME 1 → Operations 1, 2, 6 and 8 can be scheduled on a multiplier.
DG(k=multiplier, 1) = Prob (1,1) + Prob(1,2) + Prob (6, 1) + Prob (8,1) = 1.0 + 1.0 + 0.5 +0.3 = 2.8

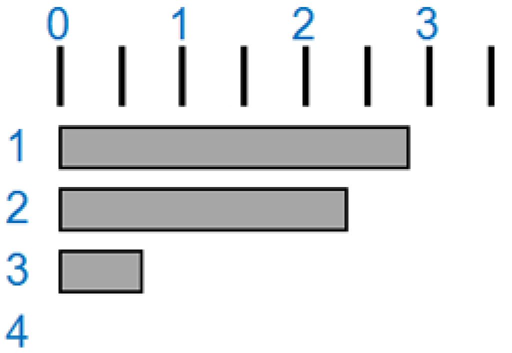
E.g., TIME 2 → Operations 3, 6, 7, and 8 can be scheduled on a multiplier.
DG(k=multiplier, 2) = Prob (3,2) + Prob(6,2) + Prob (7, 2) + Prob (8,2) = 1.0 + 0.5 + 0.5 +0.3 = 2.3

E.g., TIME 3 → Operations 7 and 8 can be scheduled on a multiplier.
DG(k=multiplier, 3) = Prob (7, 3) + Prob (8,3) = 0.5 +0.3 = 0.8

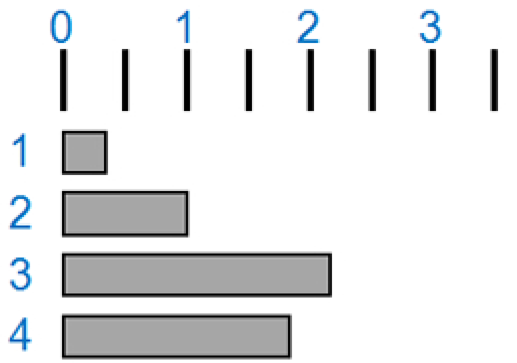
E.g., TIME 4 → No multiply operation
DG(k=multiplier, 3) = 0

Type Distribution Graph





Multiply



Add

Self Force

$$Force(i) = DG(k, i) * x(op, i)$$

DG(k, i) ~ Current Distribution Graph value

x(op, i) ~ Change in operation's probability

$$Self\ Force(j) = \quad Sum\ [Force(i)]$$

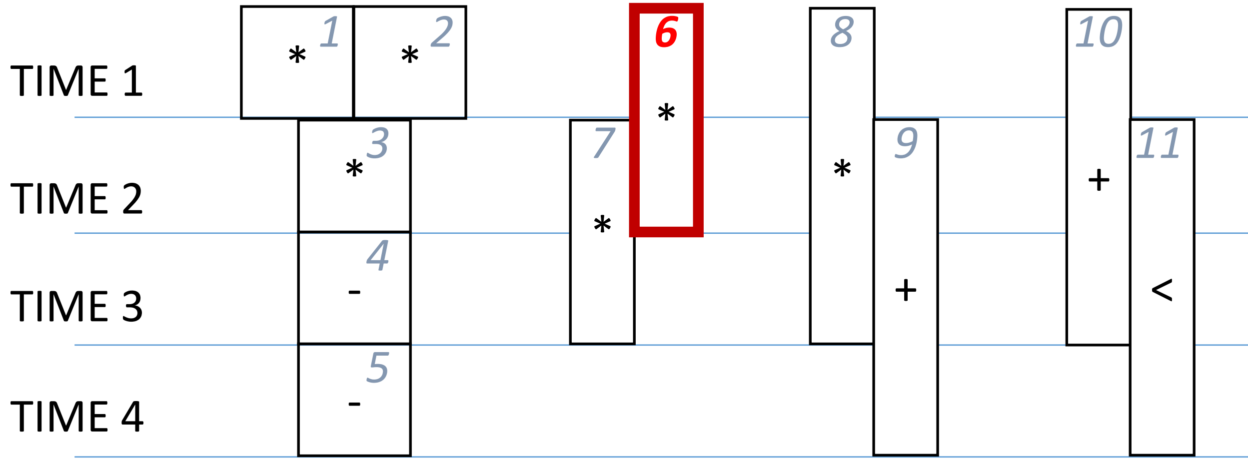
Self Force

$$Force(i) = DG(k, i) * x(op, i)$$

DG(k, i) ~ Current Distribution Graph value

x(op, i) ~ Change in operation's probability

Self Force



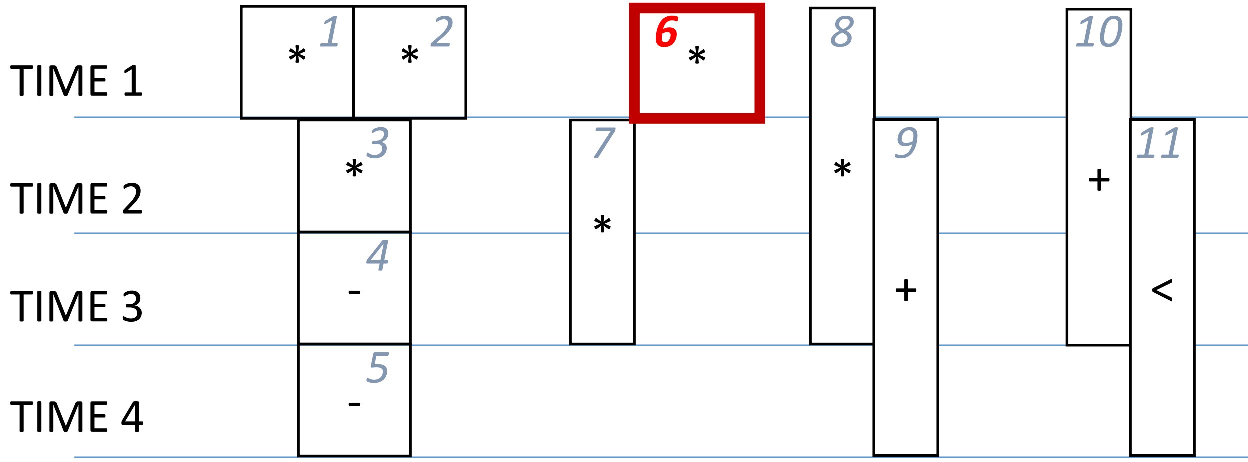
$$Force(i) = DG(k, i) * x(op, i)$$

DG(k, i) ~ Current Distribution Graph value

x(op, i) ~ Change in operation's probability

Operation 6 on TIME 1 → 0.5
Operation 6 on TIME 2 → 0.5

Self Force



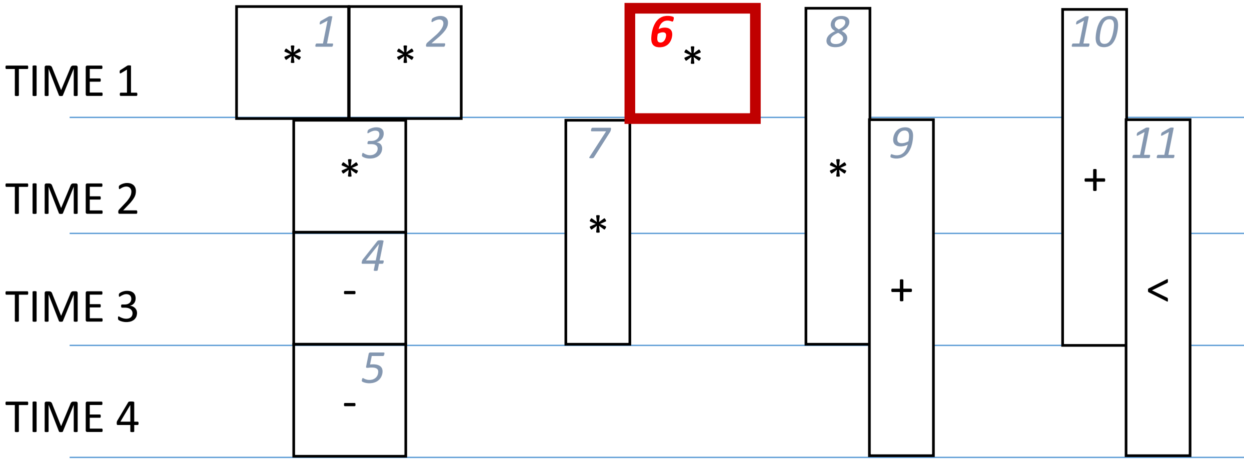
$$\text{Force}(i) = DG(k, i) * x(op, i)$$

$DG(k, i)$ ~ Current Distribution Graph value

$x(op, i)$ ~ Change in operation's probability

If Operation 6 on TIME 1 → $x(6, 1) = 1 - 0.5 = 0.5$

Self Force



$$\text{Force}(i) = DG(k, i) * x(op, i)$$

DG(k, i) ~ Current Distribution Graph value

x(op, i) ~ Change in operation's probability

If Operation 6 on TIME 1 → x(6, 1) = 1 – 0.5 = 0.5

Then Operation 6 on TIME 2 → x(6, 2) = 0 – 0.5 = -0.5

Self Force

Force(i) = DG(k, i) * x(op, i)

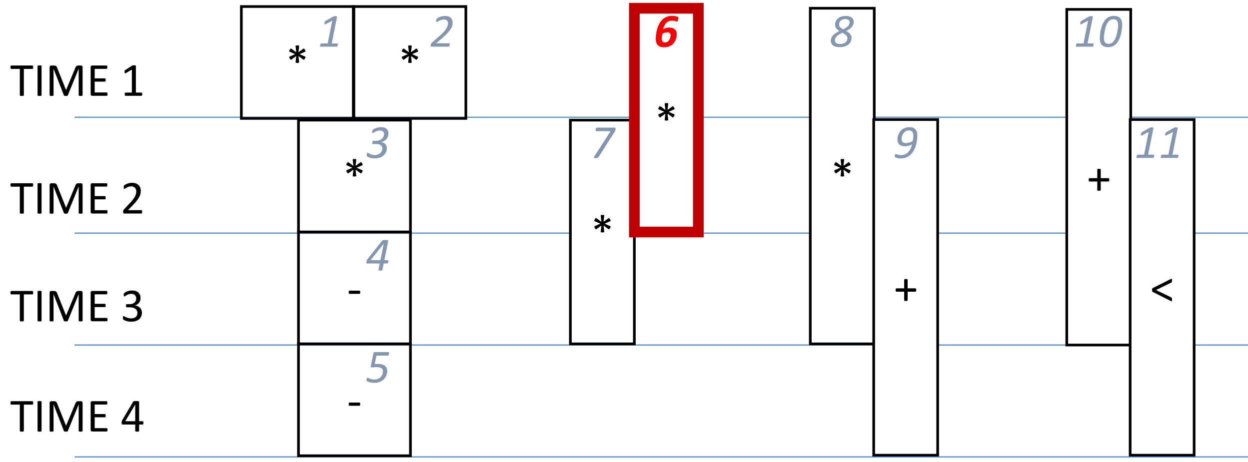
DG(k, i) ~ Current Distribution Graph value

x(op, i) ~ Change in operation’s probability

Self Force(j) = Sum [Force(i)]

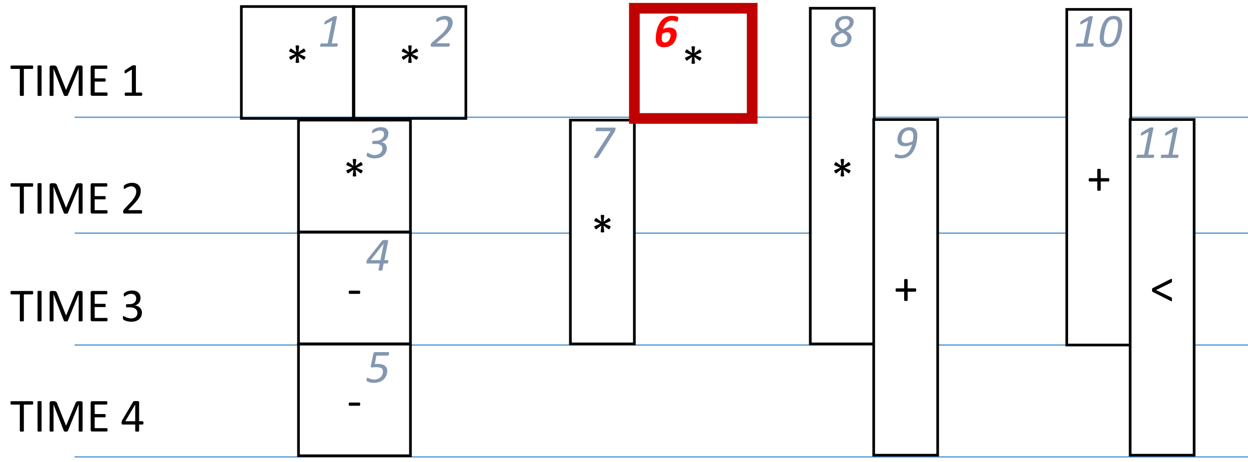
Self Force

❖Try scheduling operation 6 on TIME 1



Self Force

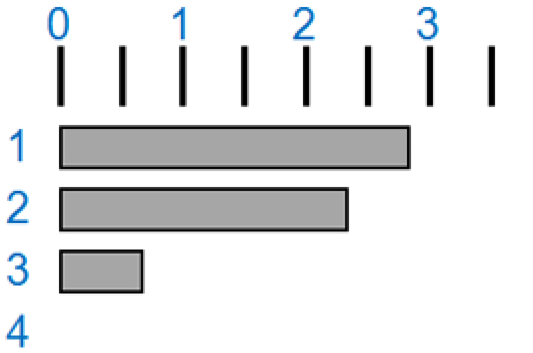
❖Try scheduling operation 6 on TIME 1



Self Force(1) = Force(1) + Force(2)

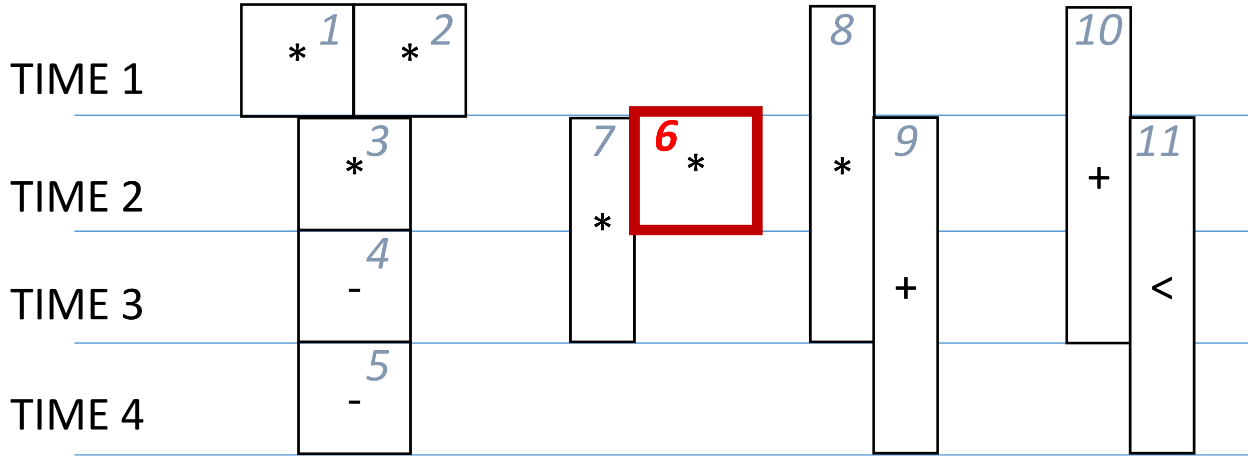
= (DG(m, 1) * X(6, 1)) + (DG(m, 2) * X(6, 2))

= [2.833*(1-0.5) + 2.333 * (0-0.5)] = +0.25



Self Force

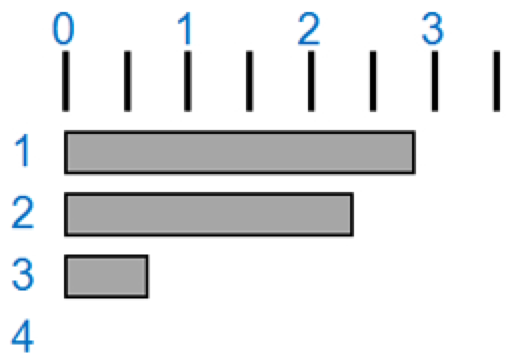
❖Try scheduling operation 6 on TIME 2



Self Force(1) = Force(1) + Force(2)

= (DG(m, 1) * X(6, 1)) + (DG(2) * X(6, 2))

= [2.833*(0-0.5) + 2.333 * (1-0.5)] = -0.25



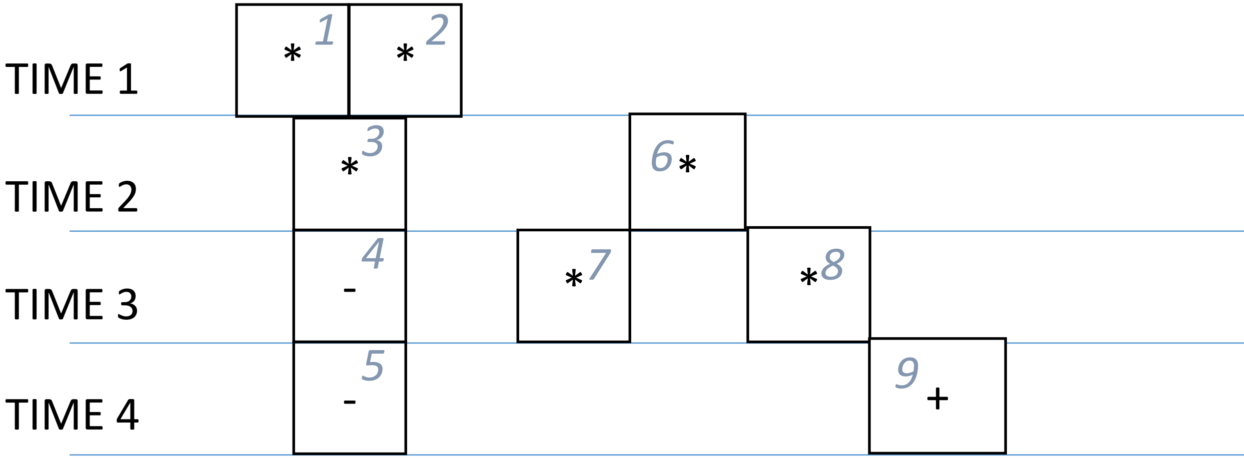
Force-Directed Scheduling

Algorithm 3: Force Directed Scheduling

Procedure `ForceDirectedScheduling()`
  Data: $G_S(V, E)$
  `/* V: Vertices, E: Edges`
  Result: t
  repeat
    Compute time frames
    Compute operation probabilities
    Compute DG
    Compute self-force, succ/pred force
    Schedule the operation with least force
  until *all operations are scheduled*;

Binding

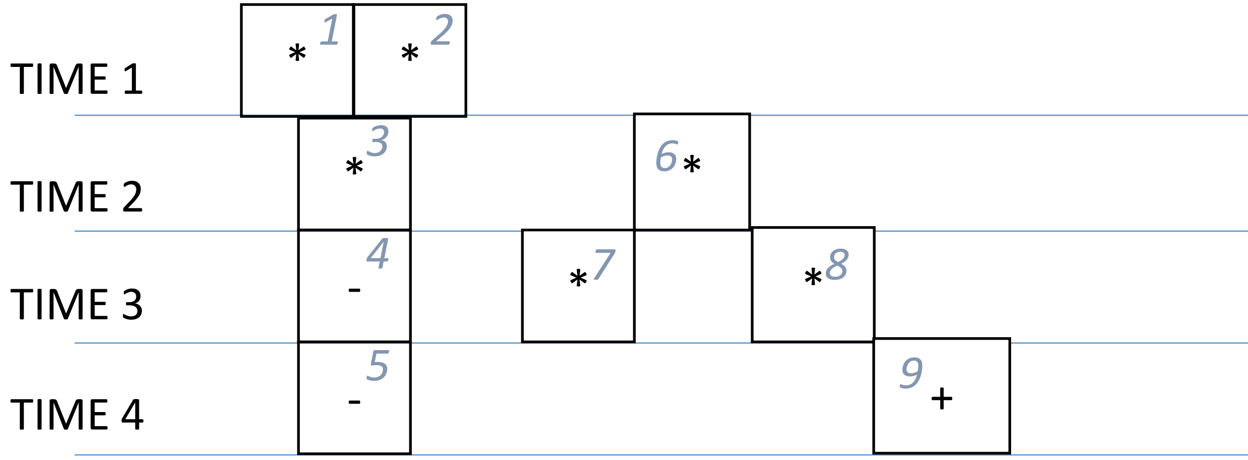
Binding (assigning) scheduled "Operations" to physical hardware resources



Given two multipliers and two address, how to bind "operations" ?

Binding

Binding (assigning) scheduled "Operations" to physical hardware resources



Mul 1 →{1, 3, 7}

Adder 1 →{4, 5}

Mul 2 → {2, 6, 8}

Adder 2 → {9}

Allocation (Register)

Allocating logical (variables) registers to physical registers

Goal: Using less physical registers under already scheduled operations

R1 → [1,4)
R2 → [4, 7)
R3 → [7, 10)
R4 → [1,9)
R5 → [8,10)
R6 → [1,3)
R7 → [3,8)

Allocation (Register)

Allocating logical (variables) registers to physical registers

Goal: Using less physical registers under already scheduled operations

R1 → [1,4)
R2 → [4, 7)
R3 → [7, 10)
R4 → [1,9)
R5 → [8,10)
R6 → [1,3)
R7 → [3,8)

How many physical registers do we need ?

Allocation (Register)

Allocating logical (variables) registers to physical registers

Goal: Using less physical registers under already scheduled operations

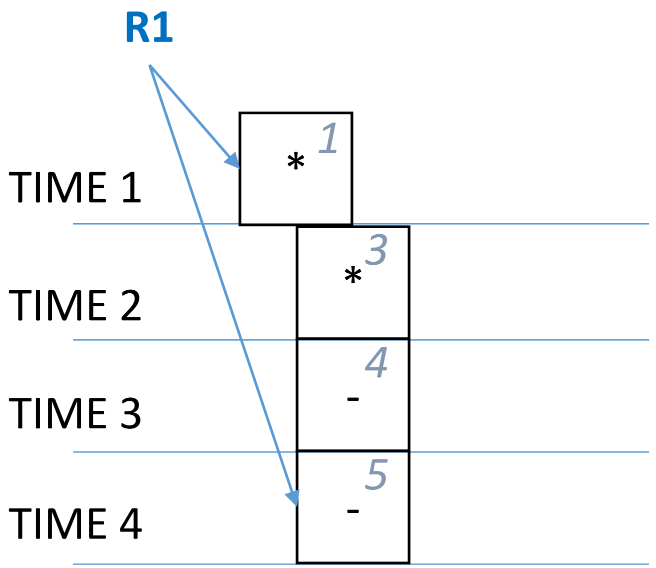
R1 → [1,4)
R2 → [4, 7)
R3 → [7, 10)
R4 → [1,9)
R5 → [8,10)
R6 → [1,3)
R7 → [3,8)

How many physical registers do we need ?

(one of the) solution(s): Left-edge algorithm

Allocation (Register): Left edge algorithm

Registers have a "life time"



Life time of **R1** starts at TIME 1 and ends at TIME 4;

OR

R1 → [1,4)

Allocation (Register): Left edge algorithm

R1 → [1,4)
R2 → [4, 7)
R3 → [7, 10)
R4 → [1,9)
R5 → [8,10)
R6 → [1,3)
R7 → [3,8)

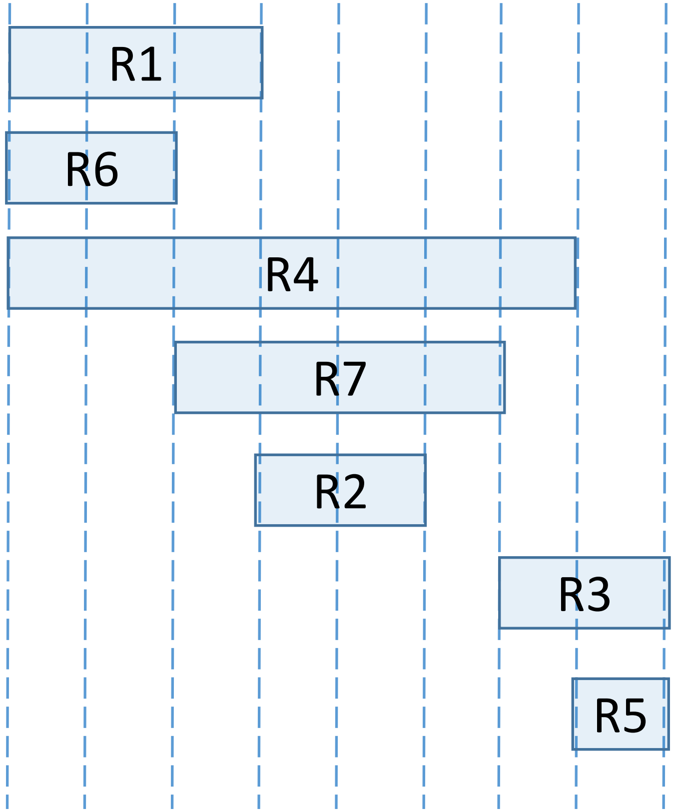
How many physical registers do we need ?

Allocation (Register): Left edge algorithm

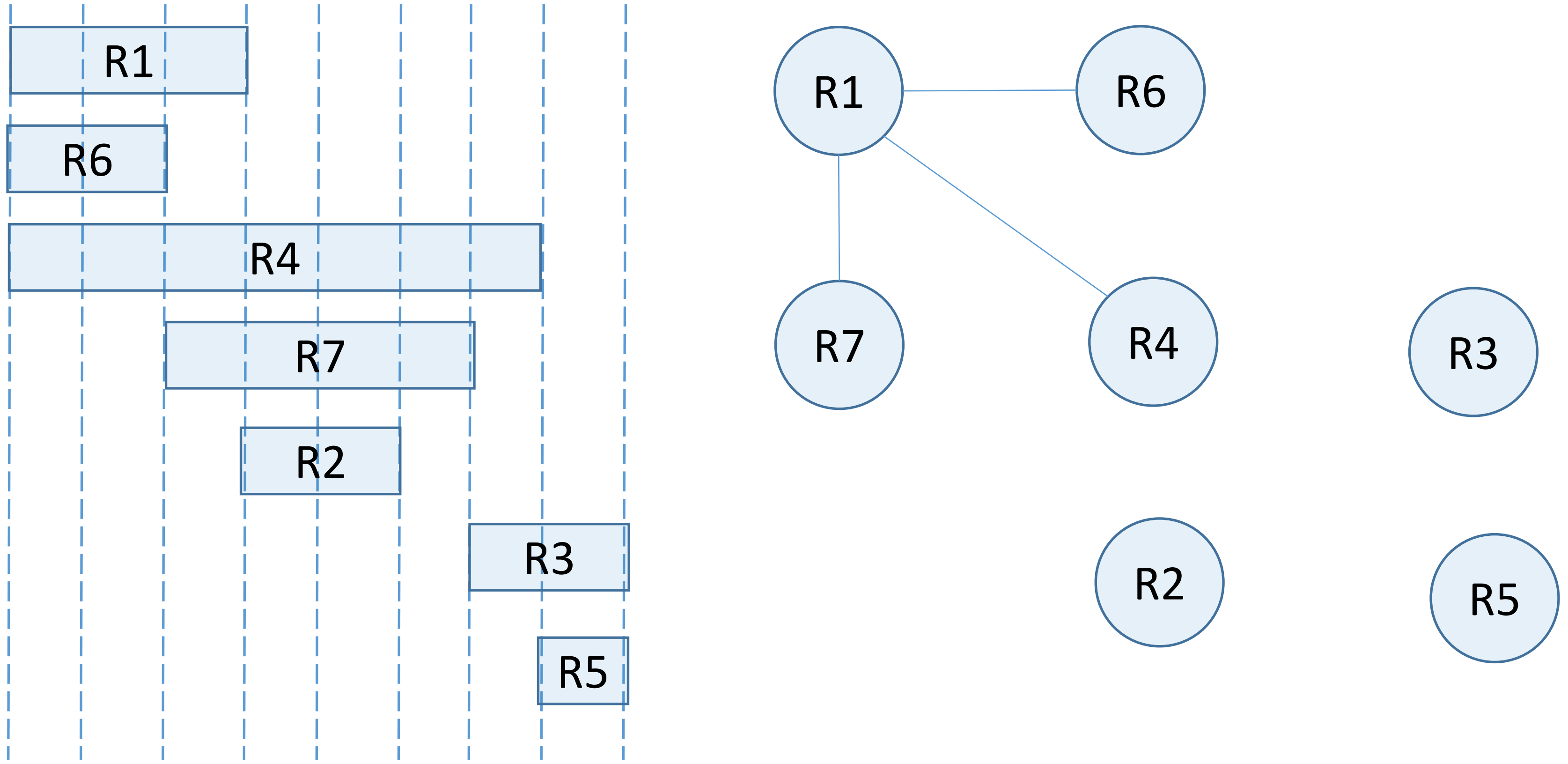
R1 → [1,4)
R2 → [4, 7)
R3 → [7, 10)
R4 → [1,9)
R5 → [8,10)
R6 → [1,3)
R7 → [3,8)

R1 → [1,4)
R6 → [1,3)
R4 → [1,9)
R7 → [3,8)
R2 → [4, 7)
R3 → [7, 10)
R5 → [8,10)

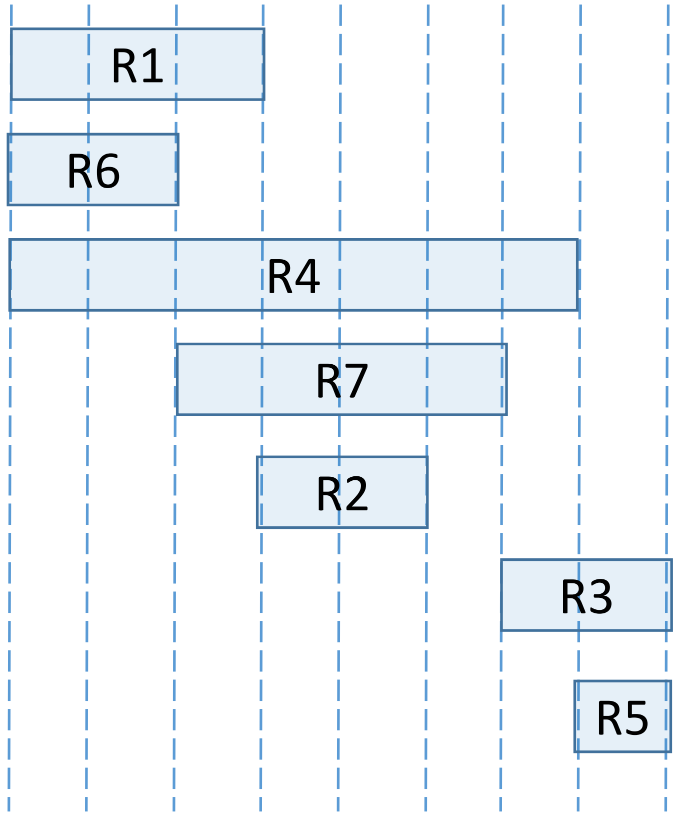
Sort

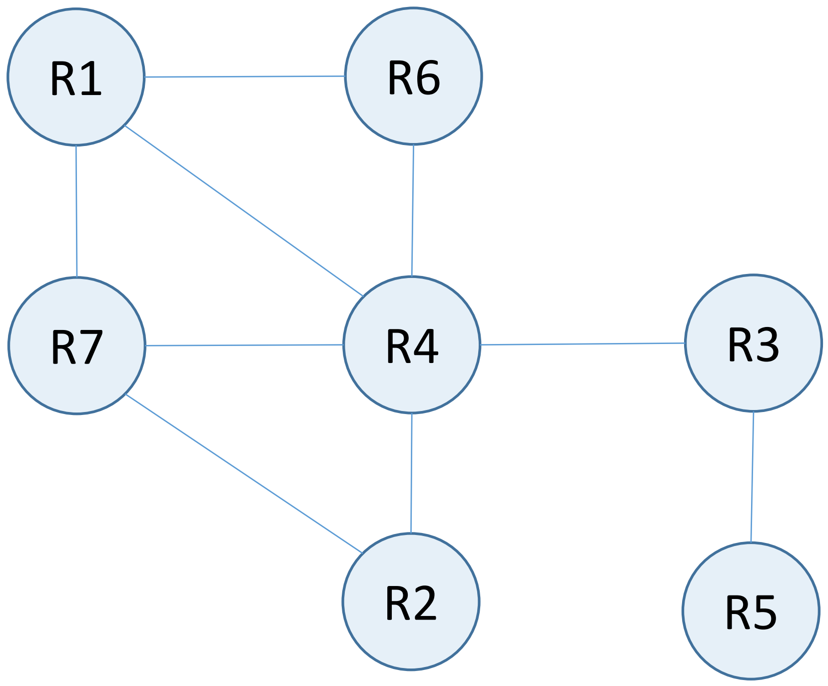


Allocation (Register): Left edge algorithm



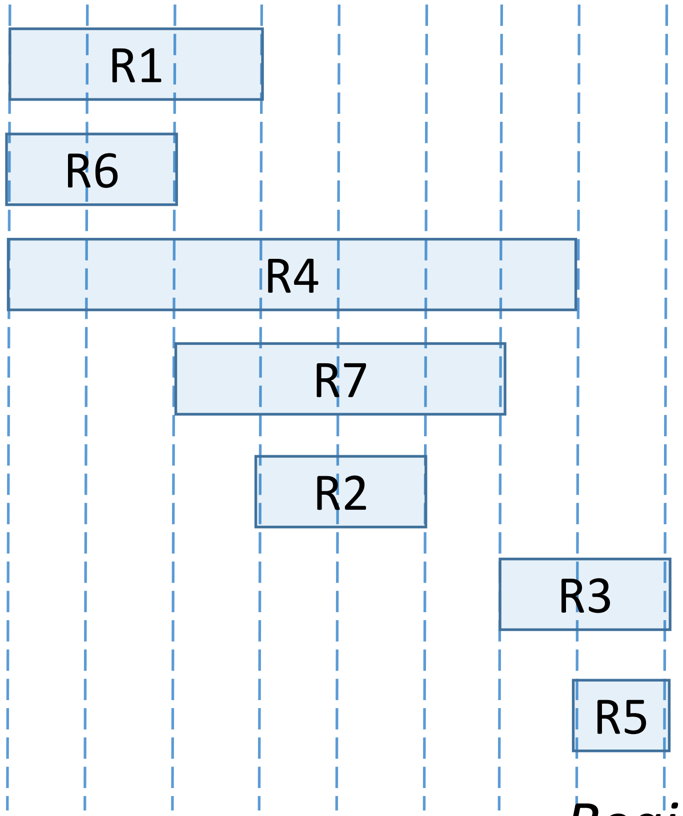
Allocation (Register): Left edge algorithm

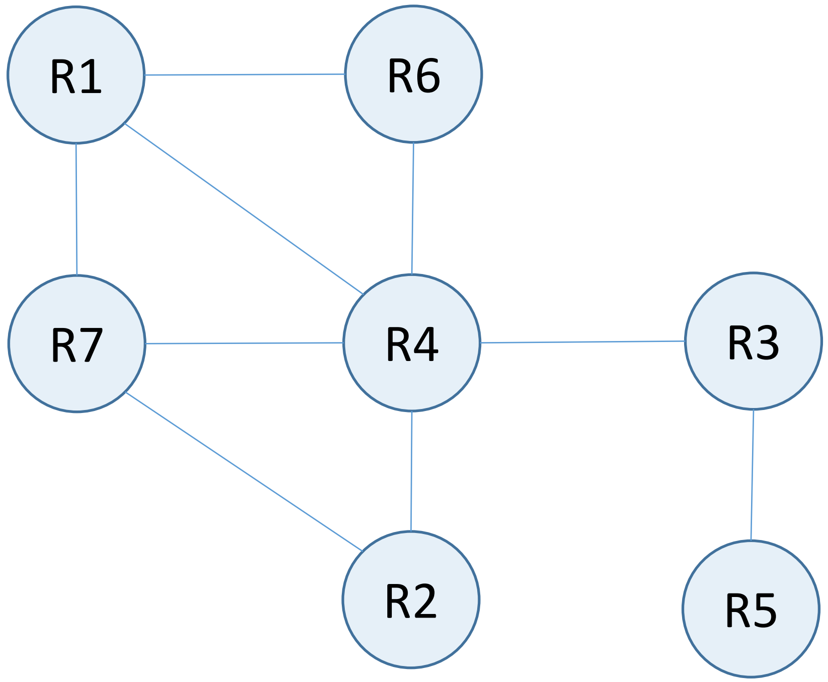




Register Allocation → Graph Coloring Problem

Allocation (Register): Left edge algorithm



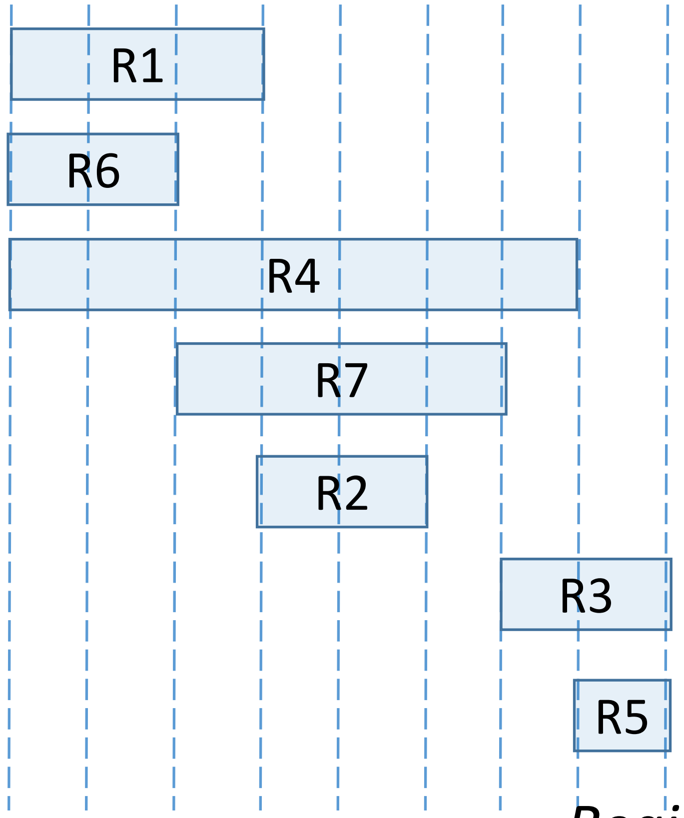


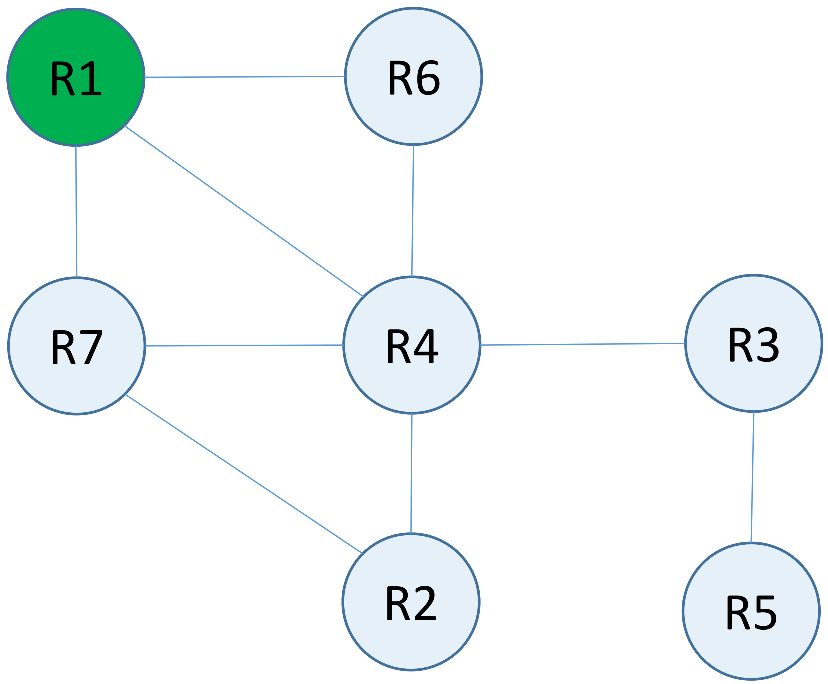
Register Allocation → Graph Coloring Problem

Graph Coloring : Color vertices such that vertices with common edge must have different colors

Goal: Use minimum number of colors

Allocation (Register): Left edge algorithm



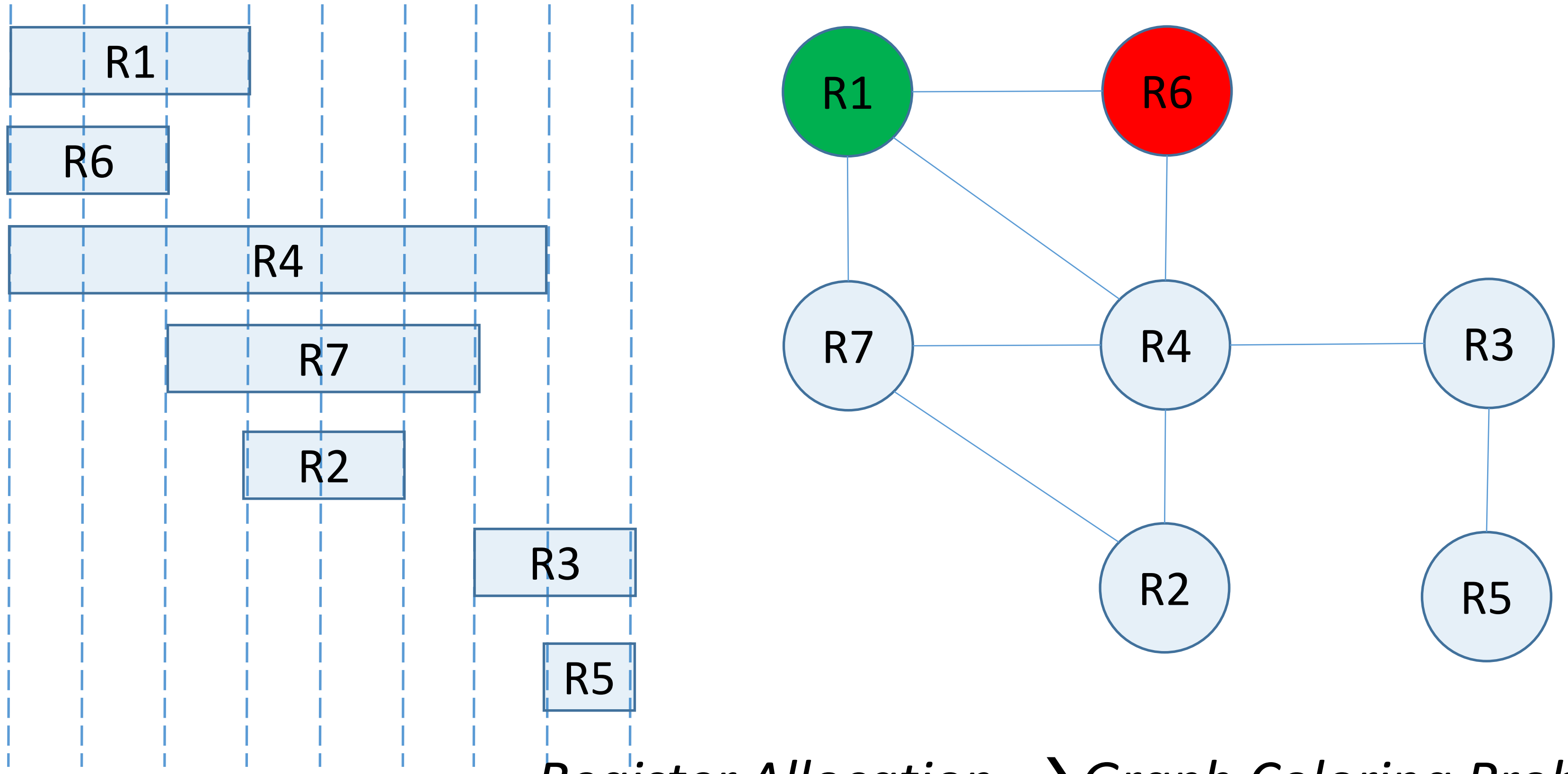


Register Allocation → Graph Coloring Problem

Graph Coloring : Color vertices such that vertices with common edge must have different colors

Goal: Use minimum number of colors

Allocation (Register): Left edge algorithm

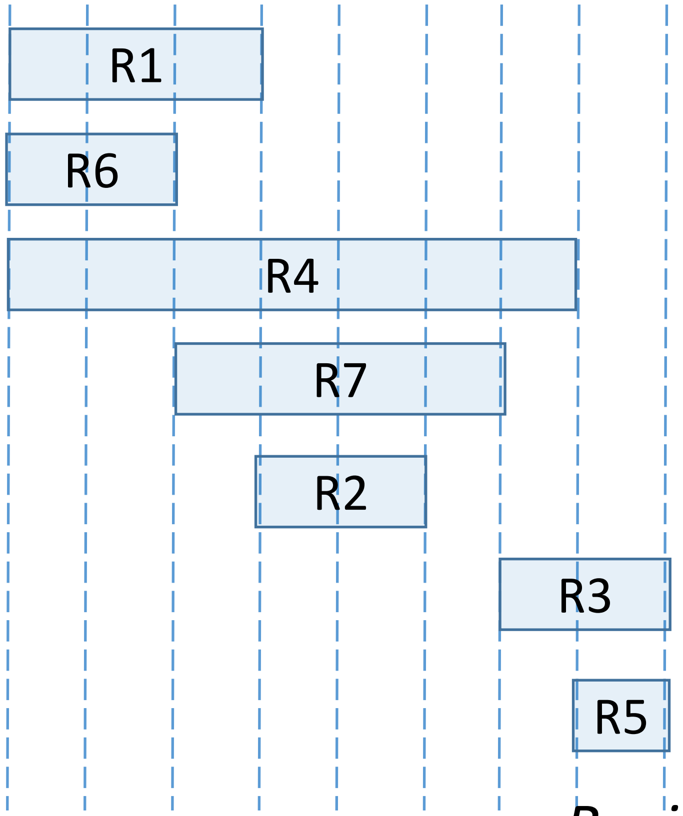


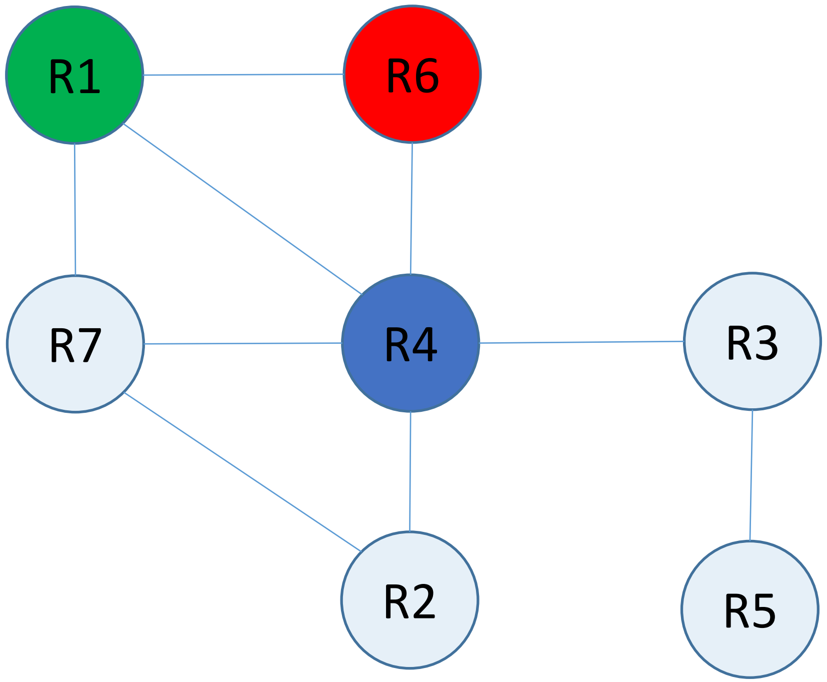
Register Allocation → Graph Coloring Problem

Graph Coloring : Color vertices such that vertices with common edge must have different colors

Goal: Use minimum number of colors

Allocation (Register): Left edge algorithm



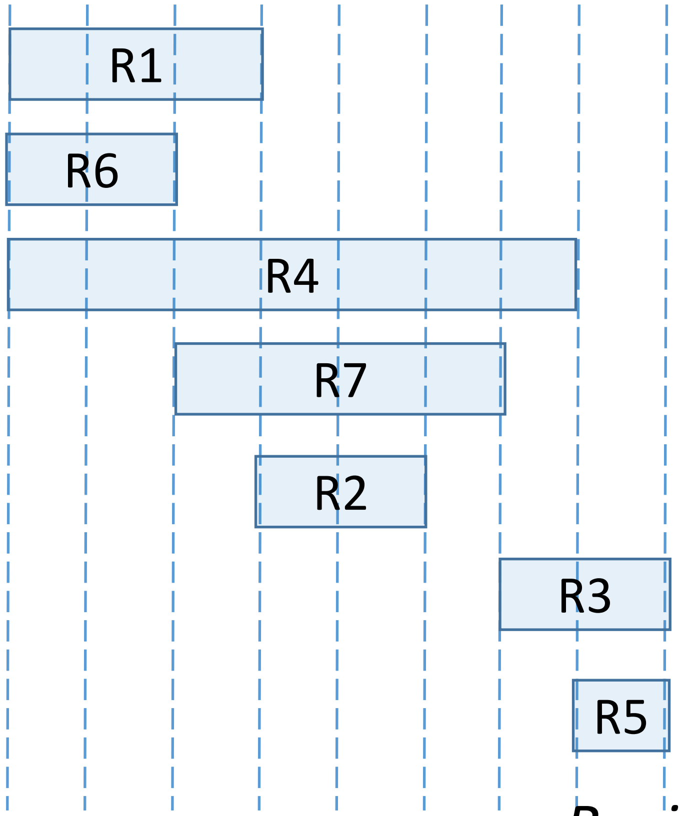


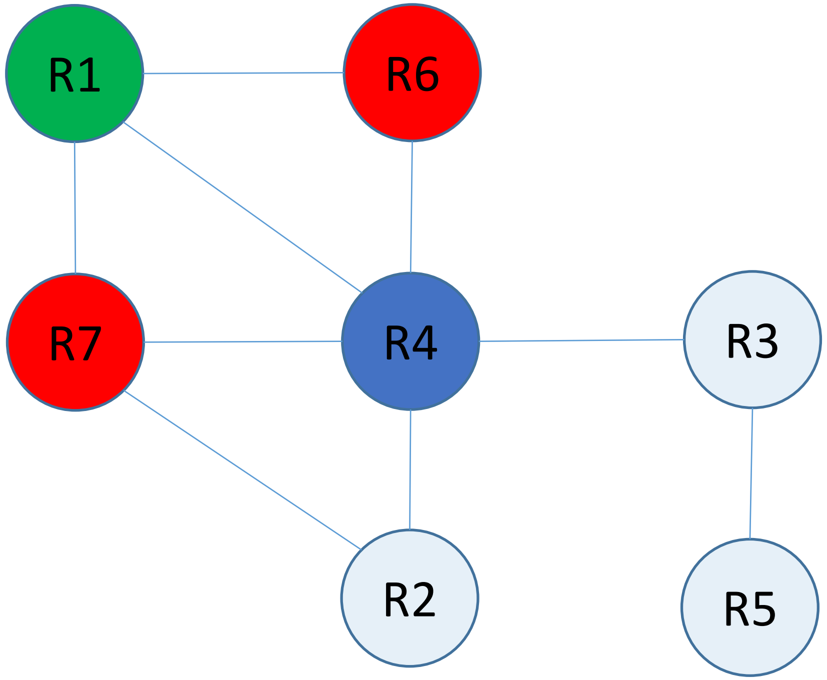
Register Allocation → Graph Coloring Problem

Graph Coloring : Color vertices such that vertices with common edge must have different colors

Goal: Use minimum number of colors

Allocation (Register): Left edge algorithm



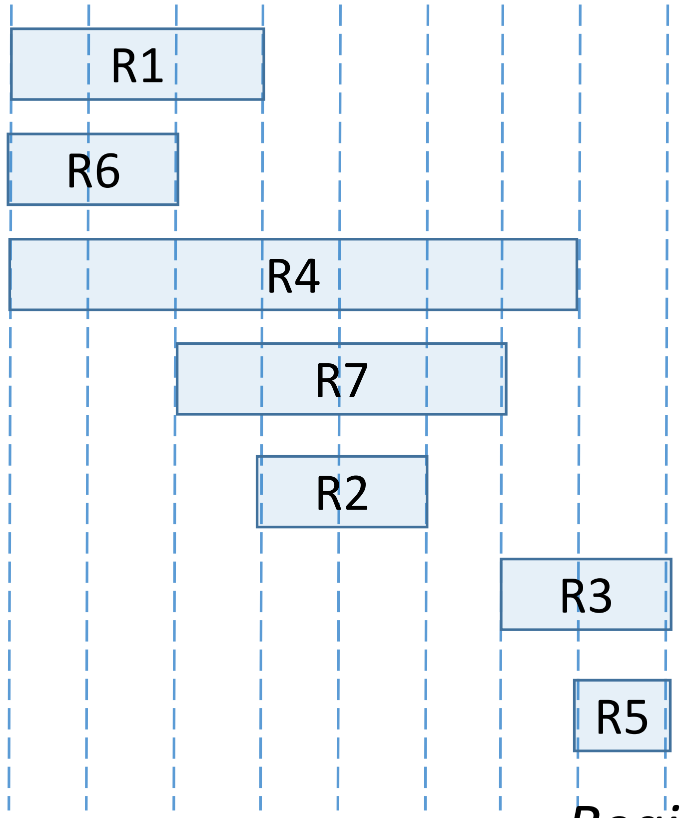


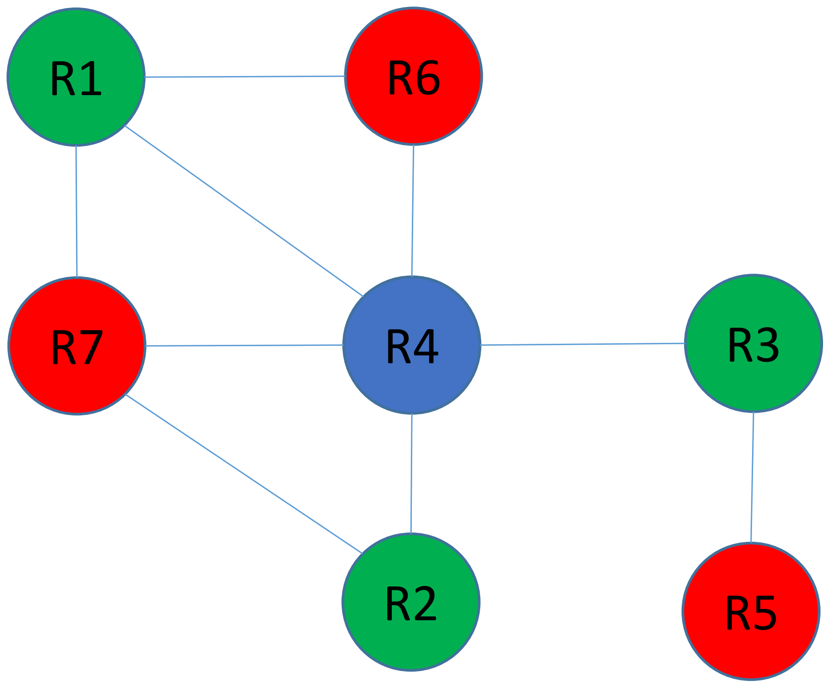
Register Allocation → Graph Coloring Problem

Graph Coloring : Color vertices such that vertices with common edge must have different colors

Goal: Use minimum number of colors

Allocation (Register): Left edge algorithm



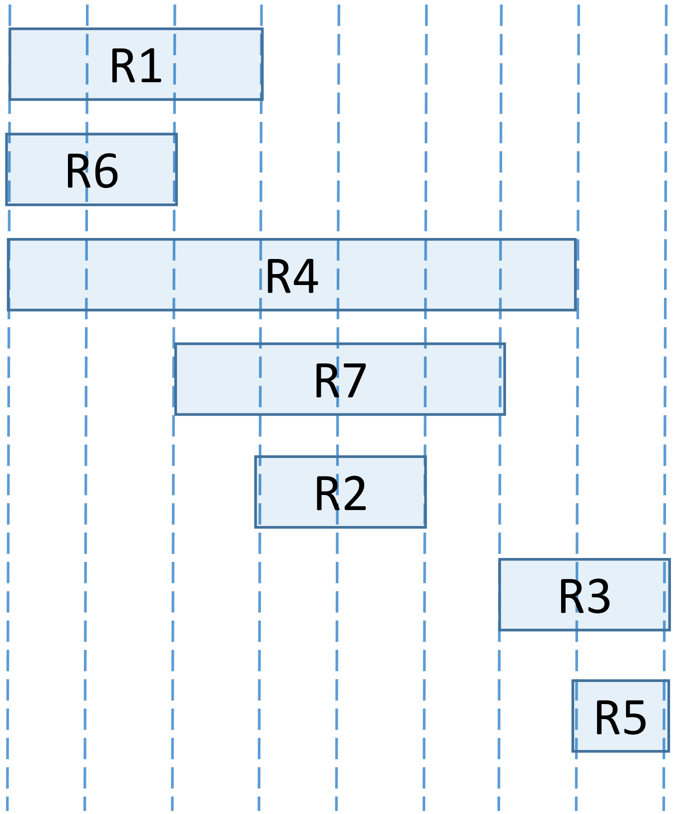


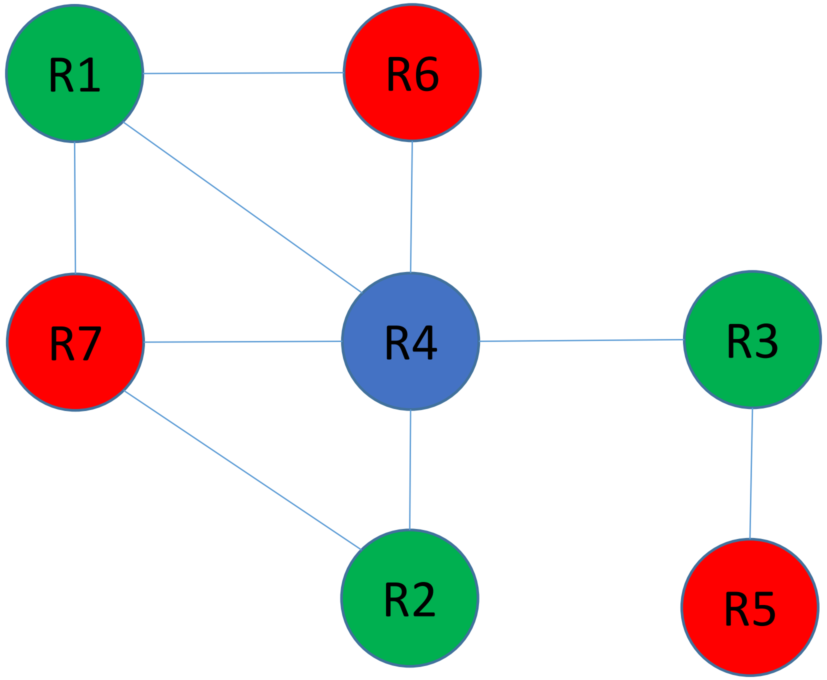
Register Allocation → Graph Coloring Problem

Graph Coloring : Color vertices such that vertices with common edge must have different colors

Goal: Use minimum number of colors

Allocation (Register): Left edge algorithm

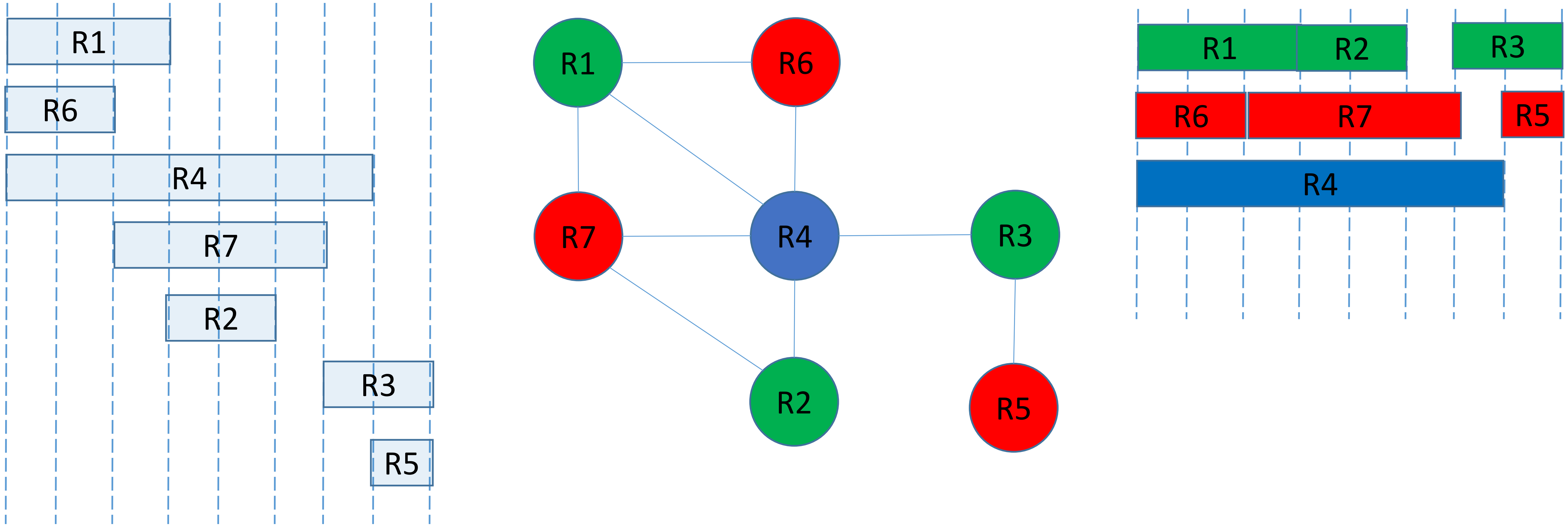




Register Allocation → Graph Coloring Problem

We need three colors (3 physical registers)

Allocation (Register): Left edge algorithm



Register Allocation → Graph Coloring Problem

We need three colors (3 physical registers)

Vivado High-Level Synthesis Demo

C-Sim, Co-Sim, Implementation, Clock period, Area, Clock Cycles, II (Initiation Interval),…

Vivado HLS Intro

- Vivado HLS GUI
- Vivado HLS TCL project creation
- Viado HLS Terminology
  - C-Sim,
  - Co-Sim,
  - Implementation,
  - Clock period, Area, Clock Cycles, II (Initiation Interval),…