Chapter 4

ARCHITECTURAL-LEVEL SYNTHESIS

© Giovanni De Micheli
Stanford University

Please read the
entire chapter 4

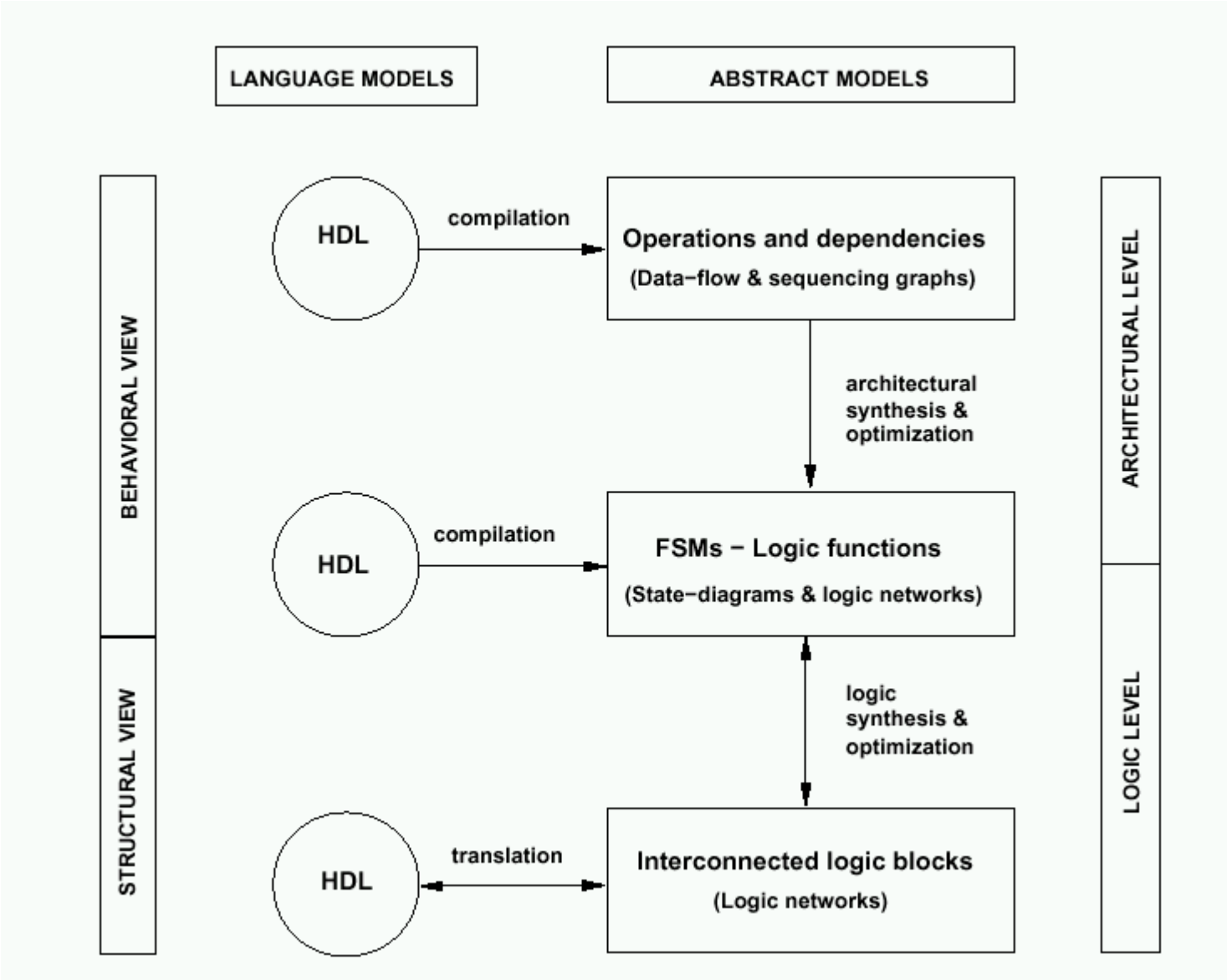
Outline

- Motivation.
- Compiling language models into abstract models.
- Behavioral-level optimization and program-level transformations.
- Architectural synthesis: an overview.

Synthesis

- Transform behavioral into structural view.
- Architectural-level synthesis:
  - Architectural abstraction level.
  - Determine ***macroscopic*** structure.
  - Example: major building blocks.
- Logic-level synthesis:
  - Logic abstraction level.
  - Determine ***microscopic*** structure.
  - Example: logic gate interconnection.

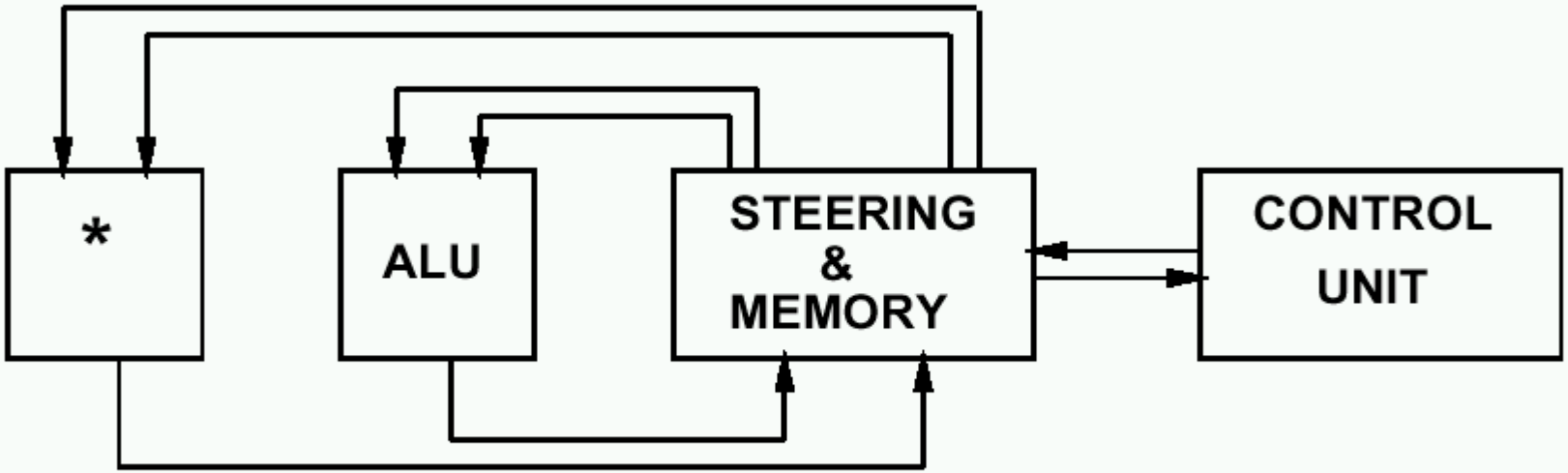
Synthesis and optimization

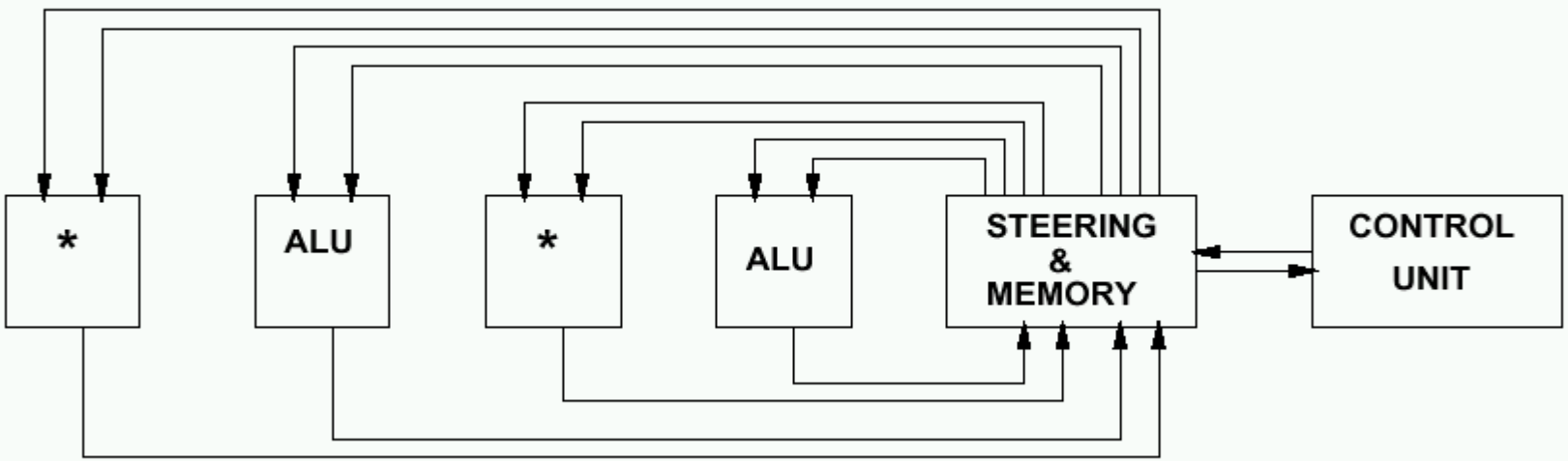


Example

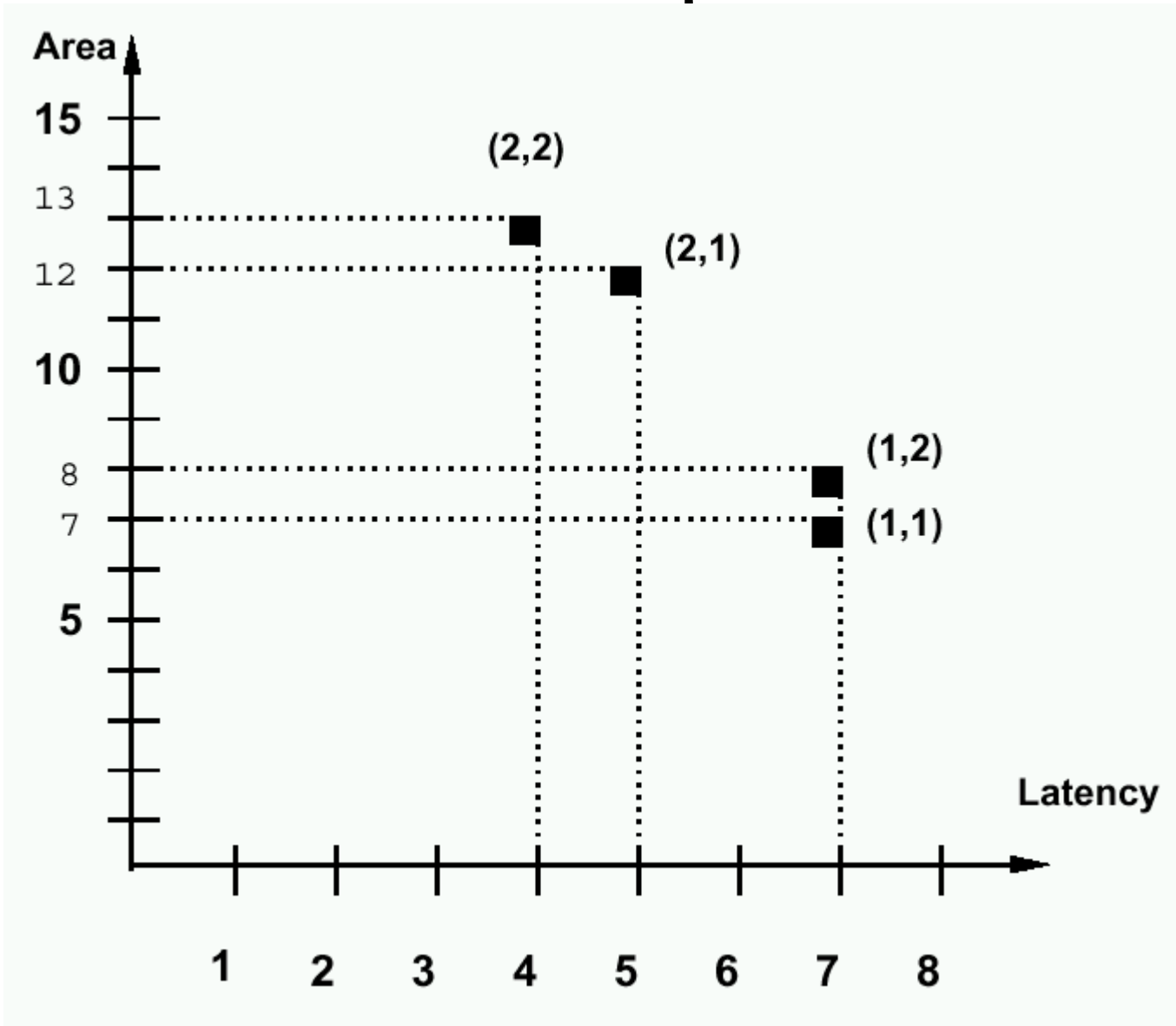
```
diffeq {
        read (x, y, u, dx, a);
        repeat {
            xl = x + dx;
            ul = u - (3 * x * u  *  dx) - (3 * y * dx);
           yl = y + u * dx;
           c = x < a;
           x = xl; u = ul; y = yl;
           }
      until ( c ) ;
      write (y);
}
```

Example of structures





Example



Architectural-level synthesis motivation

- Raise input abstraction level.
  - Reduce specification of details.
  - Extend designer base.
  - Self-documenting design specifications.
  - Ease modifications and extensions.
- Reduce design time.
- Explore and optimize macroscopic structure:
  - Series/parallel execution of operations.

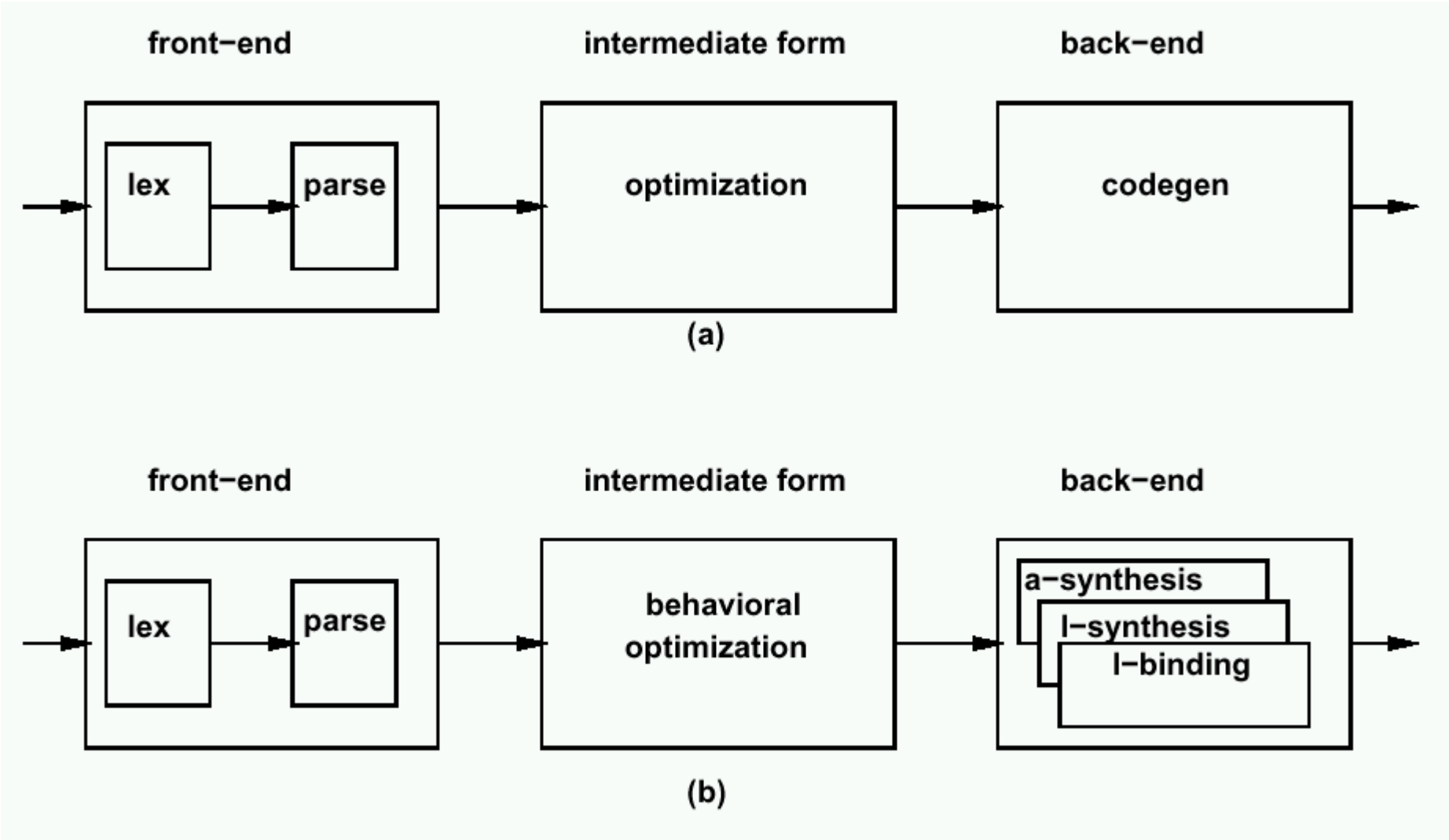
Architectural-level synthesis

- Translate HDL models into sequencing graphs.
- Behavioral-level optimization:
  - Optimize abstract models independently from the implementation parameters.
- Architectural synthesis and optimization:
  - Create macroscopic structure:
    - data-path and control-unit.
  - Consider area and delay information of the implementation.

Compilation and behavioral optimization

- Software compilation:
  - Compile program into intermediate form.
  - Optimize intermediate form.
  - Generate target code for an architecture.
- Hardware compilation:
  - Compile HDL model into sequencing graph.
  - Optimize sequencing graph.
  - Generate gate-level interconnection for a cell library.

Hardware and software compilation.

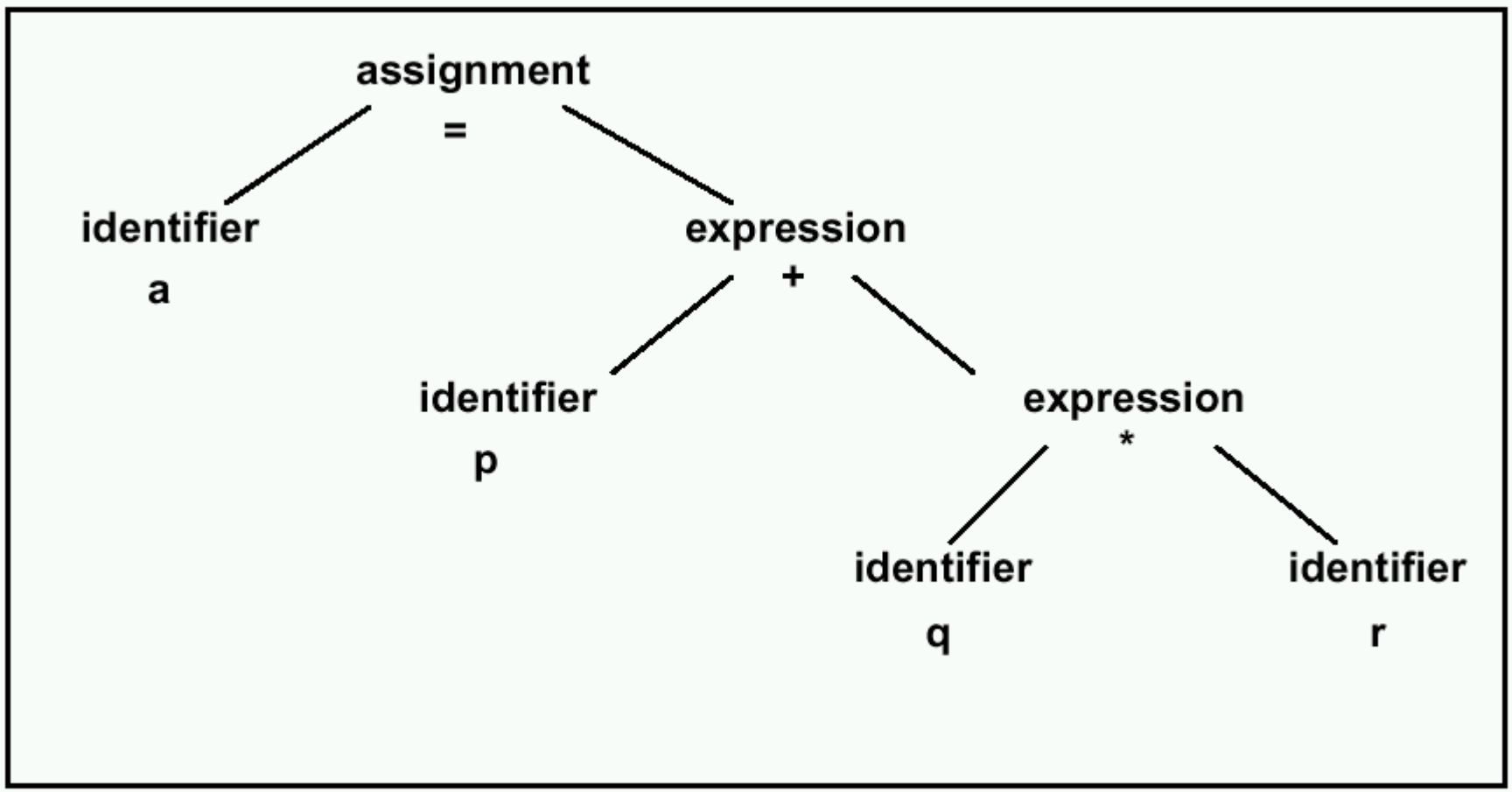


Compilation

- Front-end:
  - Lexical and syntax analysis.
  - Parse-tree generation.
  - Macro-expansion.
  - Expansion of meta-variables.
- Semantic analysis:
  - Data-flow and control-flow analysis.
  - Type checking.
  - Resolve arithmetic and relational operators.

Parse tree example

a = p +q r



Behavioral-level optimization

- Semantic-preserving transformations aiming at simplifying the model.
- Applied to parse-trees or during their generation.
- Taxonomy:
  - ***Data-flow*** based transformations.
  - ***Control-flow*** based transformations.

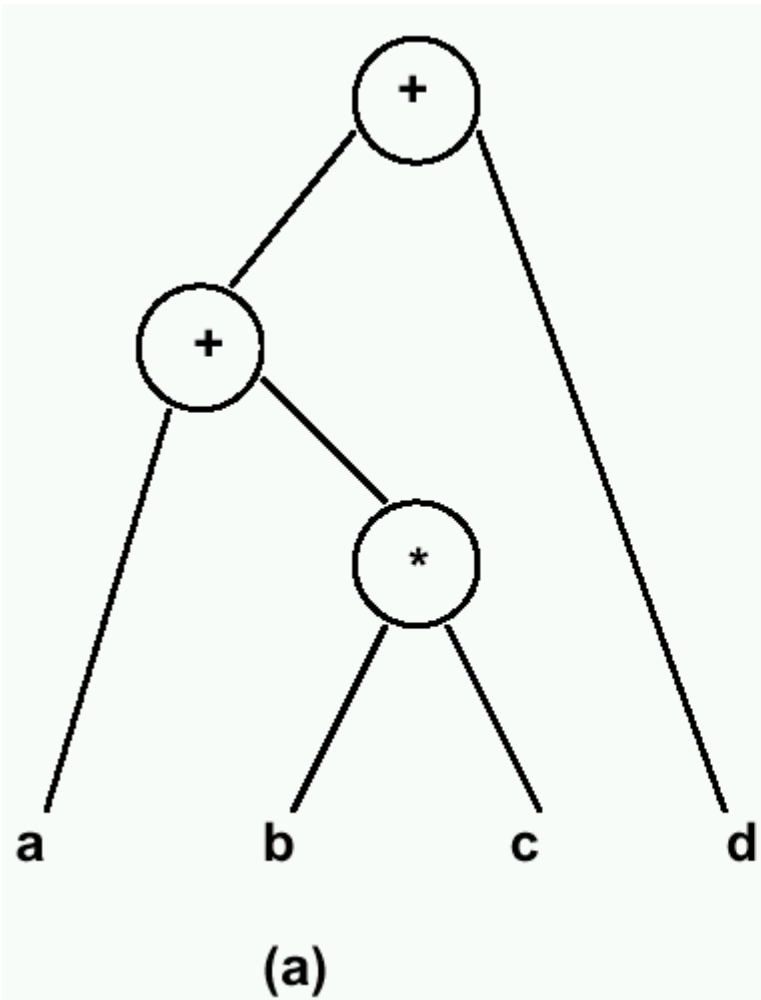
Data-flow based transformations

- Tree-height reduction.
- Constant and variable propagation.
- Common sub-expression elimination.
- Dead-code elimination.
- Operator-strength reduction.
- Code motion.

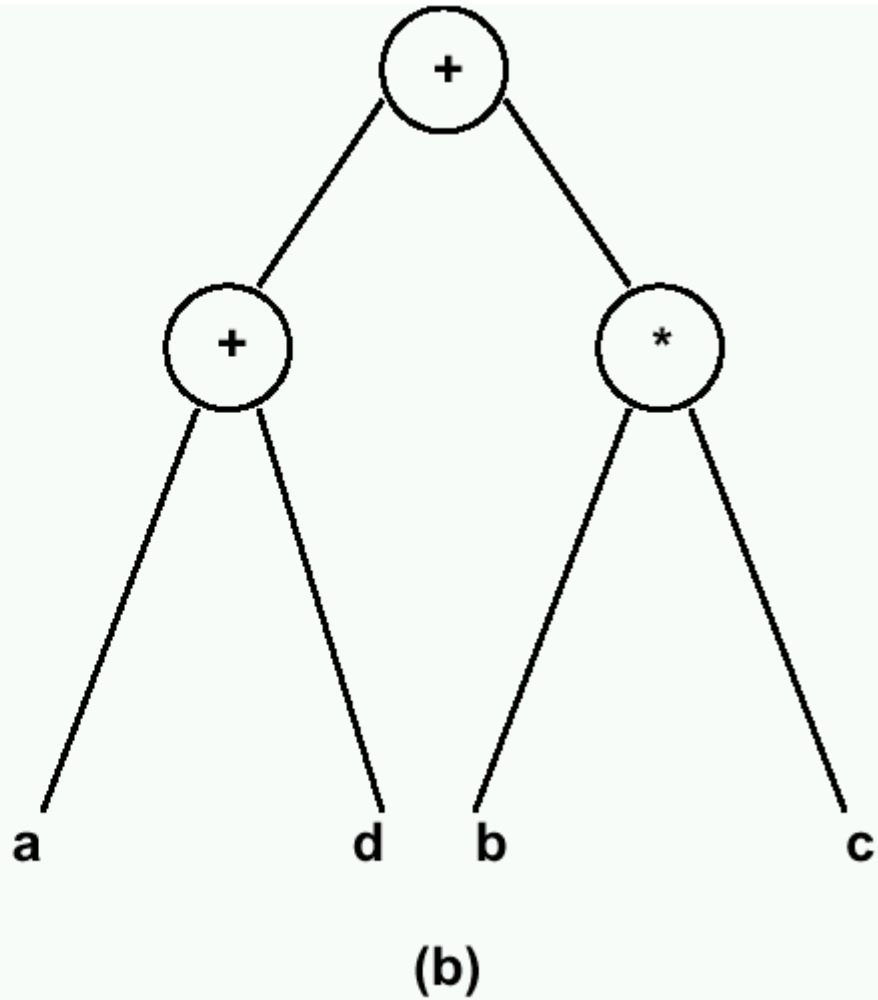
Tree-height reduction

- Applied to arithmetic expressions.
- Goal:
  - Split into two-operand expressions to exploit hardware parallelism at best.
- Techniques:
  - Balance the expression tree.
  - Exploit ***commutativity, associativity and distributivity.***

Example of tree-height reduction using commutativity and associativity



(a)

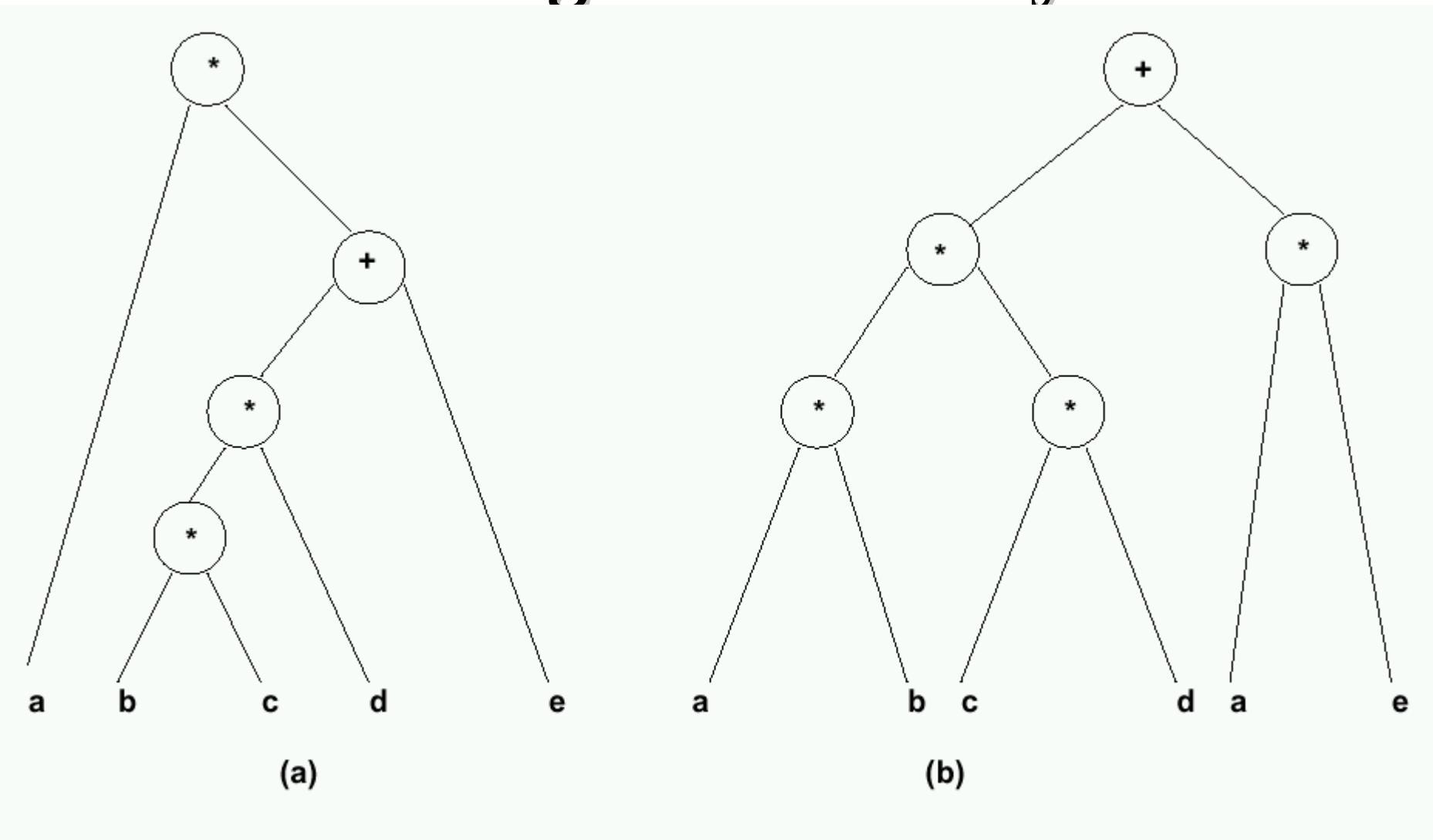


(b)

x = a +b c +d)

x = (a +d) +b c

Example of tree-height reduction using distributivity



x = a (b c d +e))

x = a b c d +a e;

Examples of propagation

- First Transformation type: Constant propagation:
  - a = 0, b = a +1, c = 2 * b,
  - a = 0, b = 1, c = 2,
- Second Transformation type: Variable propagation:
  - a = x, b = a +1, c = 2 * a,
  - a = x, b = x +1, c = 2 * x,

Sub-expression elimination

- Logic expressions:
  - Performed by logic optimization.
  - Kernel-based methods.
- Arithmetic expressions:
  - Search isomorphic patterns in the parse trees.
  - **Example:**
    - a = x +y, b = a +1, c = x +y,
    - a = x +y, b = a +1, c = a.

Examples of other transformations

- Dead-code elimination:
  - a = x; b = x +1; c = 2 * x;
  - a = x; can be removed if not referenced.
- Operator-strength reduction:
  - $a = x^2$; b = 3 * x;
  - a = x * x; t = x << 1; b = x + t;
- Code motion:
  - for (i = 1; i ≤ a * b) { }
  - t = a * b; for (i = 1; i ≤ t) { }

Control- flow based transformations

- Model expansion.
- Conditional expansion.
- Loop expansion.
- Block-level transformations.

Model expansion

- Expand subroutine flatten hierarchy.
- Useful to expand scope of other optimization techniques.
- Problematic when routine is called more than once.
- **Example:**
  - x = a +b; y = a * b; z = foo(x; y);
  - foo(p; q) {t = q - p; return(t); }
  - By expanding foo:
  - x = a +b; y = a * b; z = y - x

Conditional expansion

- Transform conditional into parallel execution with test at the end.
- Useful when test depends on late signals.
- May preclude hardware sharing.
- Always useful for logic expressions.
- Example:
  - y = ab; **if** (a) {x = b + d; } **else** {x = bd;}
  - can be expanded to: x = a(b +d) +**a'** bd
  - and simplified as: y = ab; x = y +d(a +b)

Loop expansion

- Applicable to loops with data-independent exit conditions.
- Useful to expand scope of other optimization techniques.
- Problematic when loop has many iterations.
- Example:
  - x = 0;

    for (i = 1; i ≤ 3; i ++) {x = x +1; }
  - Expanded to:

    x = 0; x = x +1; x = x +2; x = x +3

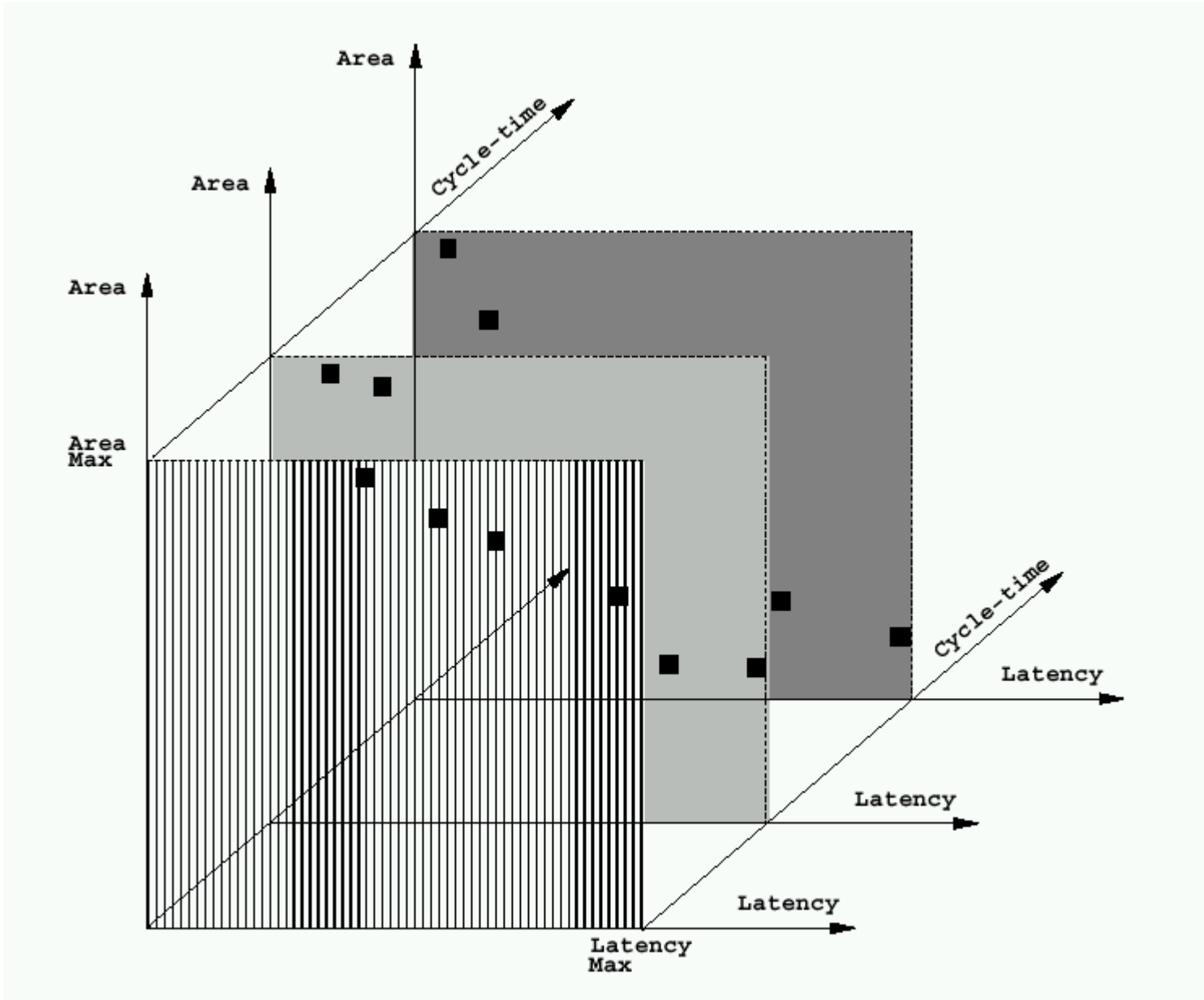
Architectural synthesis and optimization

- Synthesize macroscopic structure in terms of building-blocks.
- Explore area/performance trade-o:
  - maximum performance implementations subject to area constraints.
  - minimum area implementations subject to *performance constraints*.
- Determine an optimal implementation.
- Create logic model for data-path and control.

Design space and objectives

- Design space:
  - Set of all feasible implementations.
- Implementation parameters:
  - Area.
  - *Performance:*
    - Cycle-time.
    - Latency.
    - Throughput (for pipelined implementations).
  - Power consumption

Design evaluation space



Hardware modeling

- Circuit behavior:
  - Sequencing graphs.
- Building blocks:
  - Resources.
- Constraints:
  - Timing and resource usage.

Resources

- Functional resources:
  - Perform operations on data.
  - Example: arithmetic and logic blocks.
- Memory resources:
  - Store data.
  - Example: memory and registers.
- Interface resources:
  - Example: busses and ports.

Functional resources

- Standard resources:
  - Existing macro-cells.
  - Well characterized (area/delay).
  - Example: adders, multipliers, ...
- Application-specific resources:
  - Circuits for specific tasks.
  - Yet to be synthesized.
  - Example: instruction decoder.

Resources and circuit families

- Resource-dominated circuits.
  - Area and performance depend on few, well-characterized blocks.
  - Example: DSP circuits.
- Non resource-dominated circuits.
  - Area and performance are strongly influenced by sparse logic, control and wiring.
  - Example: some ASIC circuits.

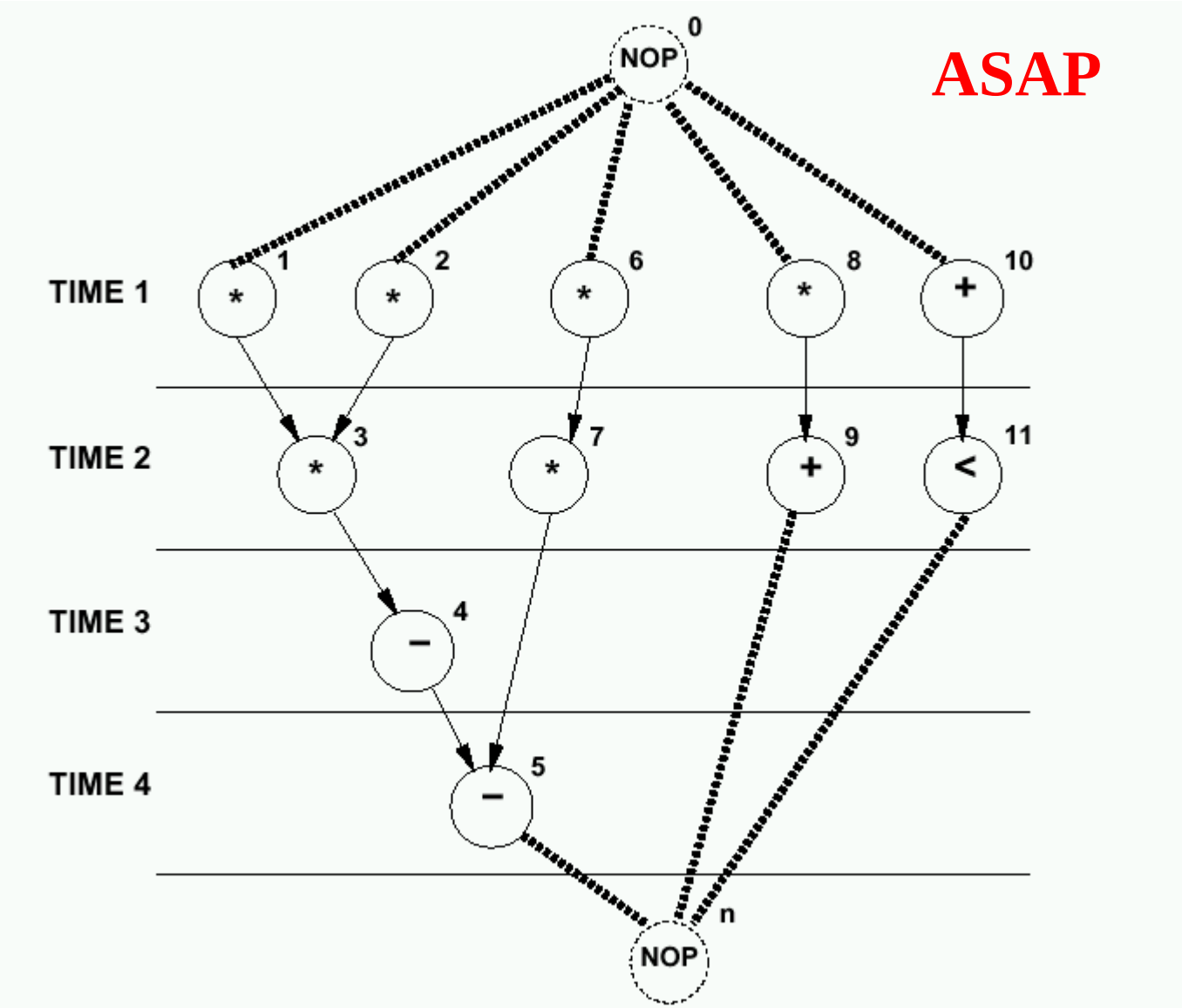
Implementation constraints

- Timing constraints:
  - Cycle-time.
  - Latency of a set of operations.
  - Time spacing between operation pairs.
- Resource constraints:
  - Resource usage (or allocation).
  - Partial binding.

Synthesis in the temporal domain

- Scheduling:
  - Associate a start-time with each operation.
  - Determine latency and parallelism of the implementation.
- Scheduled sequencing graph:
  - Sequencing graph with start-time annotation.

Example of Synthesis in the temporal domain

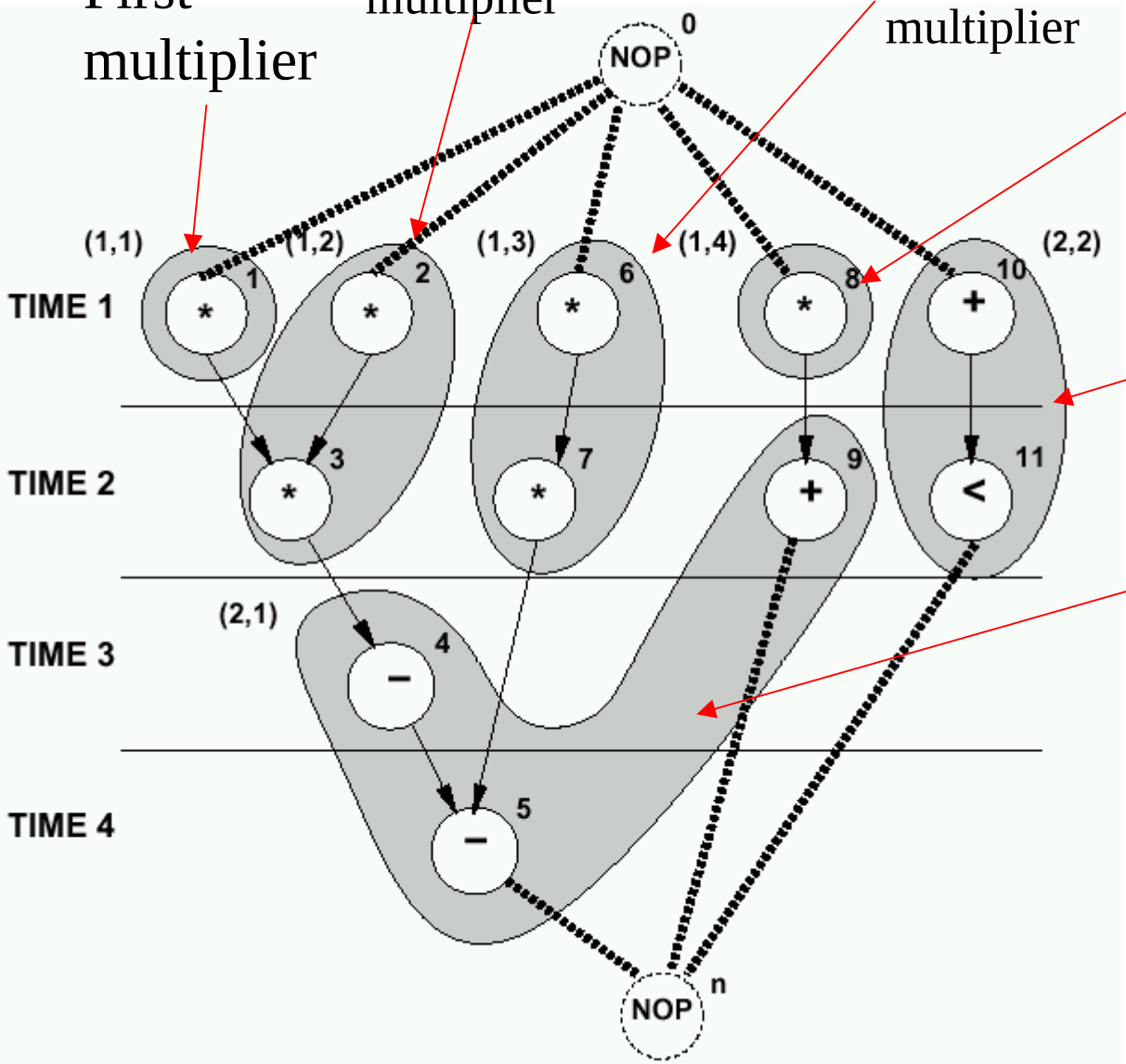


Synthesis in the spatial domain

- Binding:
  - Associate a resource with each operation with the same type.
  - Determine area of the implementation.
- Sharing:
  - Bind a resource to more than one operation.
  - Operations must not execute concurrently.
- Bound sequencing graph:
  - Sequencing graph with resource annotation.

Example of Synthesis in the spatial domain

- First multiplier
- Second multiplier
- Third multiplier
- Fourth multiplier
- First ALU
- Second ALU

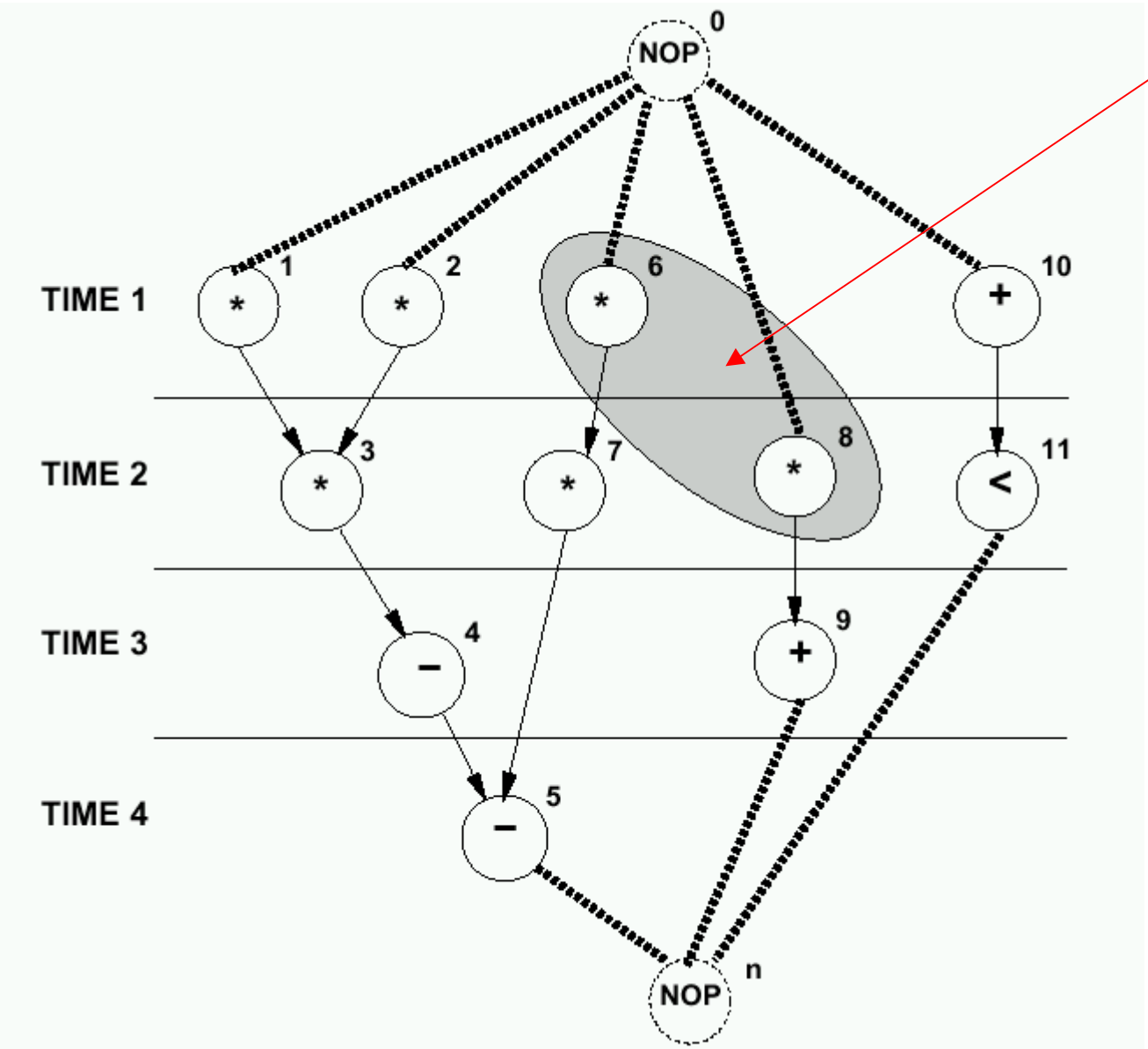


- Solution
- Four Multipliers
- Two ALUs
- Four Cycles

Binding specification

- Mapping from the vertex set to the set of resource instances, for each given type.
- Partial binding:
  - Partial mapping, given as design constraint.
- Compatible binding:
  - Binding satisfying the constraints of the partial binding.

Example of Binding specification



- Binding to the same multiplier

Estimation

- Resource-dominated circuits.
  - Area = sum of the area of the resources bound to the operations.
    - Determined by binding.
  - Latency = start time of the sink operation (minus start time of the source operation).
    - Determined by scheduling
- Non resource-dominated circuits.
  - Area also affected by:
    - registers, steering logic, wiring and control.
  - Cycle-time also affected by:
    - steering logic, wiring and (possibly) control.

Approaches to architectural optimization

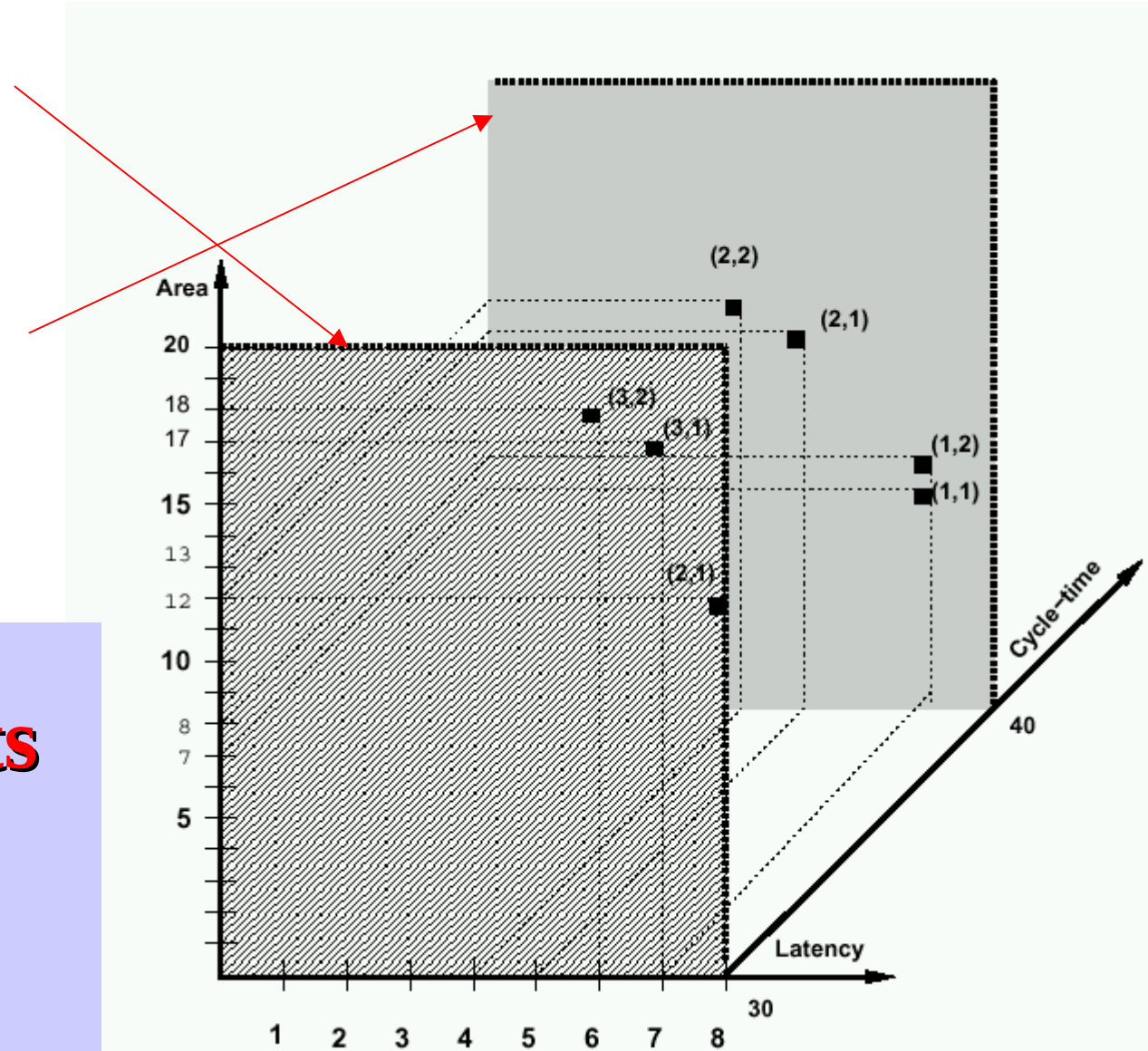
- *Multiple-criteria* optimization problem:
  - area, latency, cycle-time.
- Determine *Pareto optimal* points:
  - Implementations such that no other has all parameters with inferior values.
- Draw trade-off curves:
  - discontinuous and highly nonlinear.

Approaches to architectural optimization

- Area/latency trade-off,
  - for some values of the cycle-time.
- Cycle-time/latency trade-off,
  - for some binding (area).
- Area/cycle-time trade-off,
  - for some schedules (latency).

Area/latency trade-off for various cycle times

- Area/Latency for cycle time=30
- Area/Latency for cycle time=40



Pareto points in three dimensions

Area-latency trade-off

- Rationale:
  - Cycle-time dictated by system constraints.
- Resource-dominated circuits:
  - Area is determined by resource usage.
- Approaches:
  - Schedule for minimum latency under resource constraints
  - Schedule for minimum resource usage under latency constraints
    - for varying constraints.

Summary

- Behavioral optimization:
  - Create abstract models from HDL models.
  - Optimize models without considering implementation parameters.
- Architectural synthesis and optimization.
  - Consider resource parameters.
  - Multiple-criteria optimization problem:
    - area, latency, cycle-time.