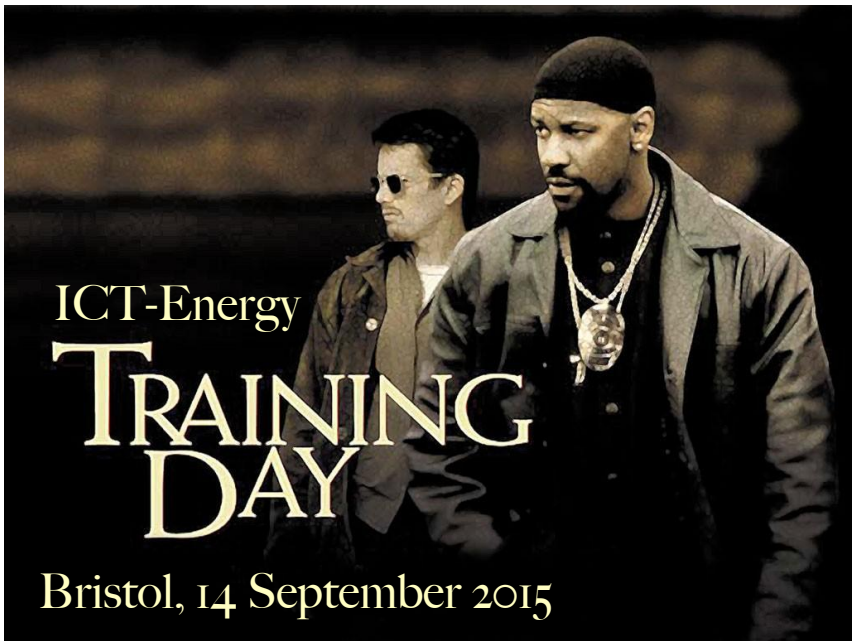
Whole-Systems Energy Transparency



John Gallagher
Roskilde University

Acknowledgements

The partners in the EU ENTRA project

Kerstin Eder and team

Pedro López García and team

Henk Muller and team

Roskilde team

entraproject.eu

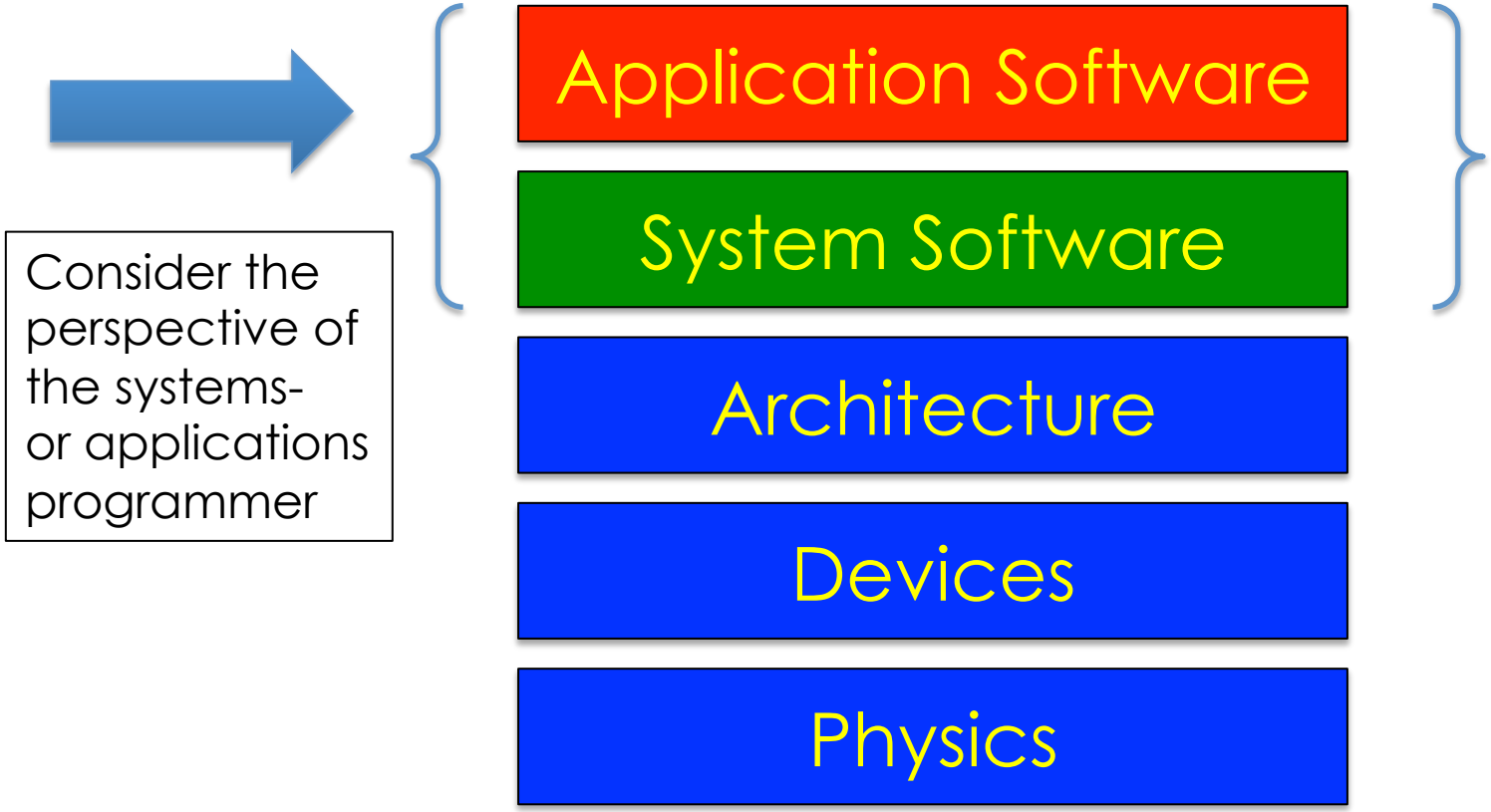
ENTRA

Energy transparency

- What does energy transparency mean?
- What is "whole-systems" energy transparency?
- How can it influence the design of low-energy ICT systems?

ENTRA

ICT-Energy and the System Stack



Consider the perspective of the systems- or applications programmer

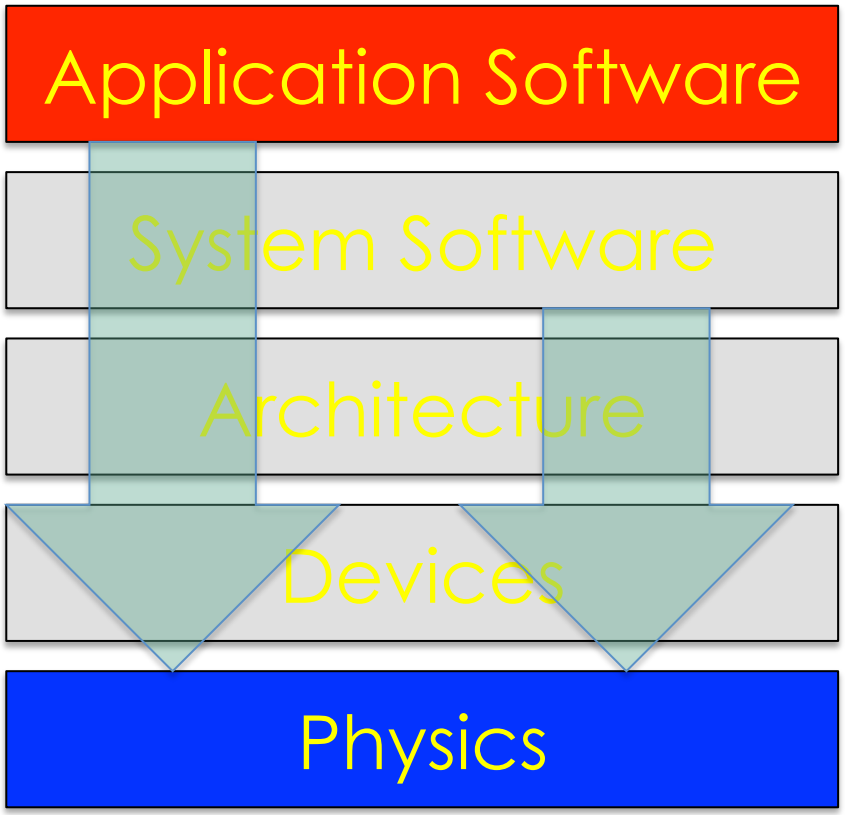
Whole-systems energy transparency

Energy is consumed by **physical processes**.

Yet, application programmers should be able to "see" through the layers and understand energy consumption at the level of code.

The same goes for designers at every level.

How is this possible?



ENTRA

Why worry about energy of *software*?

- Energy is consumed by hardware
- Hardware is getting more and more energy-efficient

- So why worry about energy-efficiency at the software level?

ENTRA

Reason 1

- Something like driving an energy-efficient car badly
- Energy-aware SW development is important (see talk from Bristol)

ENTRA

Energy waste

- Energy is wasted in ICT systems
- But where?
- Whose responsibility to fix the leaks?

ENTRA

Reason 2

- Energy efficiency as a design goal from the start
- Get an energy profile for a program as early as possible
- Analyse the code to find out how much energy a program will use
- Deliver software with energy guarantees

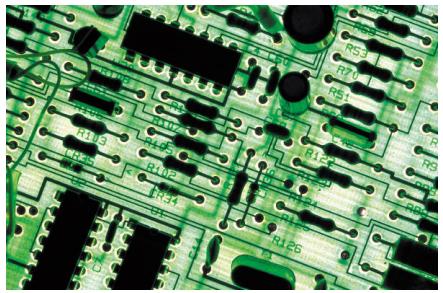
ENTRA

Reason 2 - continued

- Don't wait to test energy efficiency on hardware, after the software is developed

Development machine

Deployment platform



- It might be too late to fix "energy bugs"

ENTRA

Reason 3

- You can save more energy at the software level than the hardware level

- There are more energy optimisation opportunities higher up the system stack.
- Most energy is wasted by application software

ENTRA

Energy transparency

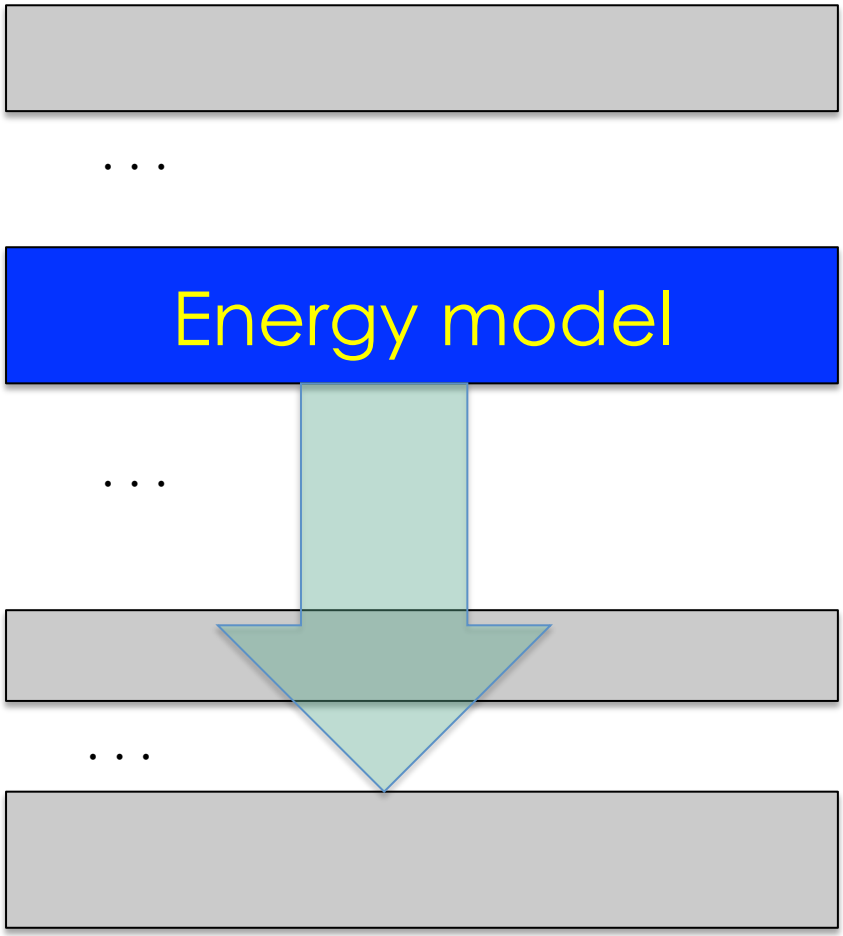
- Our aim is to let designers "see" the energy usage when working in higher levels of the system stack
  - without executing code or hardware
  - so that the programmer can see where the design wastes energy

ENTRA

Energy transparency at different levels

We need energy transparency at the level at which we are designing.

E.g. when designing an energy-efficient processor, we need an energy model of basic hardware blocks.



ENTRA

In this lecture...

- We will look at what is needed to achieve energy transparency
  - energy models and how to construct them in general
  - linking static and dynamic analysis to energy models
  - what can energy transparency give us and what are its limits?

ENTRA

Components of energy transparency

- Energy modelling
  - at different levels of the system stack
- Analysis of systems
  - to relate the energy model to a complex, dynamic entity such as a program

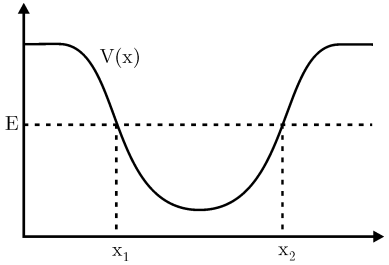
ENTRA

Energy models

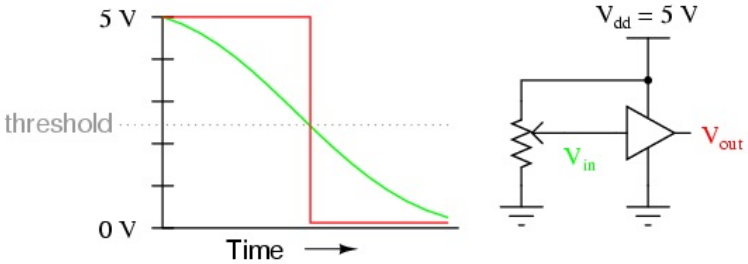
- What is an energy model?
- Typically:
  - energy costs for basic operations or components at different levels.

ENTRA

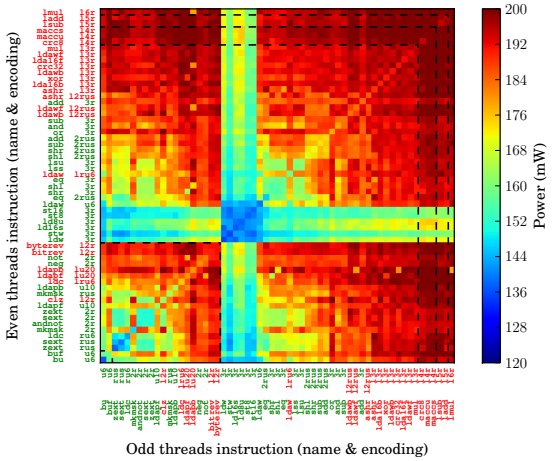
Energy models take many forms

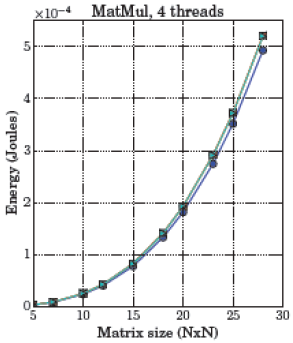


Typical response of a logic gate to a variable (analog) input voltage



Instruction	Value
Default	53.0×10^{-3} mW
zext_rus	22.0×10^{-3} mW
sext_rus	22.3×10^{-3} mW
eq_2rus	35.5×10^{-3} mW
andnot_2r	37.6×10^{-3} mW
sext_2r	37.6×10^{-3} mW
zext_2r	37.6×10^{-3} mW
mkmsk_rus	44.0×10^{-3} mW
clz_l2r	49.8×10^{-3} mW
eq_3r	51.6×10^{-3} mW
lsu_3r	52.6×10^{-3} mW
lss_3r	52.8×10^{-3} mW
shl_2rus	54.6×10^{-3} mW
add_2rus	54.8×10^{-3} mW
sub_2rus	55.7×10^{-3} mW
shr_2rus	56.6×10^{-3} mW
mkmsk_2r	60.0×10^{-3} mW
shl_3r	62.1×10^{-3} mW
shr_3r	62.2×10^{-3} mW
neg_2r	64.5×10^{-3} mW





Energy models, cont'd

- Common to all energy models –
- They provide a reference for the energy consumed by basic operations or components, at a given level of abstraction.
- E.g. energy models for
  - transistor
  - gate
  - circuit
  - machine instruction
  - JVM operation
  - C++ library call

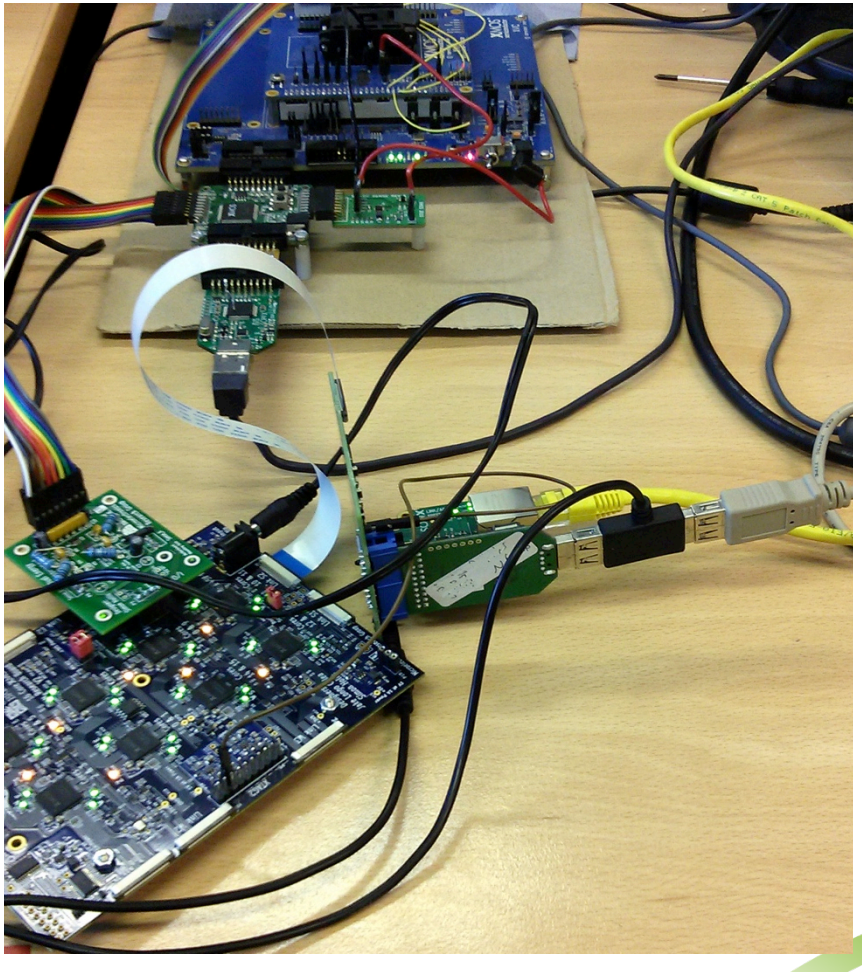
How are energy models built?

- Two basic approaches for a component
  - Empirical approach
  - Analytical approach

ENTRA

Empirical

- Measure energy consumption of components at the chosen level of abstraction
- Requires careful design of benchmarks and test harness



ENTRA

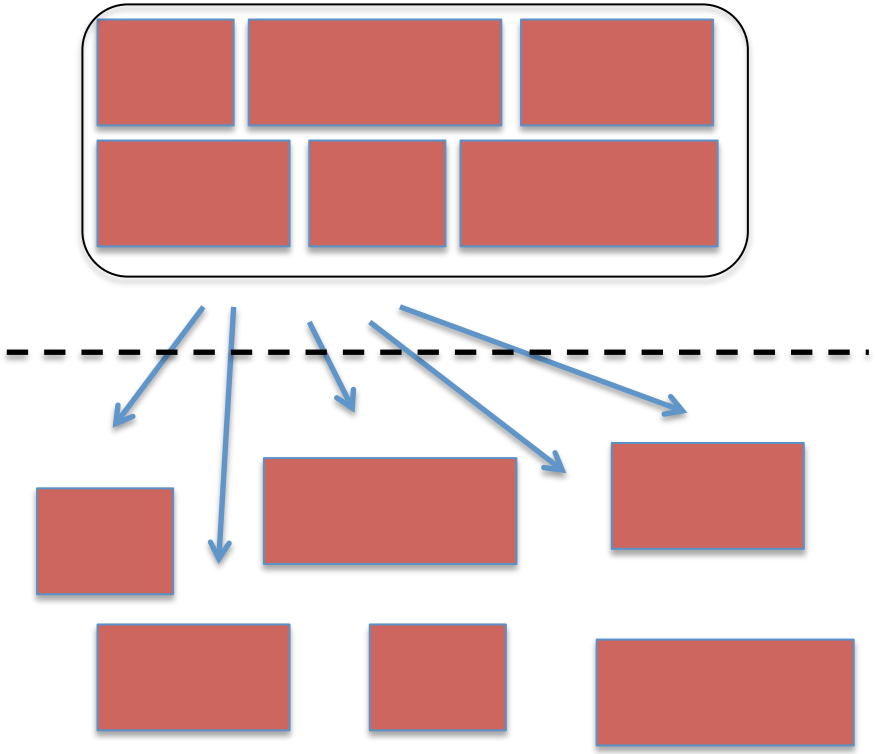
Empirical (cont'd)

- However components can seldom be measured independently
- Need to infer energy of individual operations, e.g. by
  - regression analysis
    - inferring cost of individual components from complex combinations of them
  - machine learning
    - discovering relationships between individual costs

ENTRA

Analytic (reductionist) construction

- When components at one level have a defined relationship to simpler components in lower levels
  - E.g. source code is translated to machine code
  - A circuit is constructed from gates, flip-flops etc.

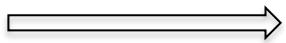


ENTRA

Analytic approach (cont'd)

- Hence compose model of a component at one level from the energy of its components at a lower level

x = (-b + sqrt(b*b – 4*a*c))/2*a



- --- 2
* --- 4
+ --- 1
/ --- 1
sqrt --- 1

An (approximate) energy model for expression evaluation could be derived from the number of operations in the expression.

ENTRA

Static analyis

- To apply an energy model is not always straightforward
- We need to understand in detail how the object under design relates to its components. E.g.
  - how often is each component called
  - what is the rate of execution of components,
  - etc.
- Then we can apply the model to achieve energy transparency

ENTRA

Energy analysis of programs

- Case study: We apply these ideas at one level of the stack
- Estimating the energy usage of an imperative program (e.g. written in C).

- We assume an energy model of the basic statements such as expression evaluation, comparison and assignment.

ENTRA

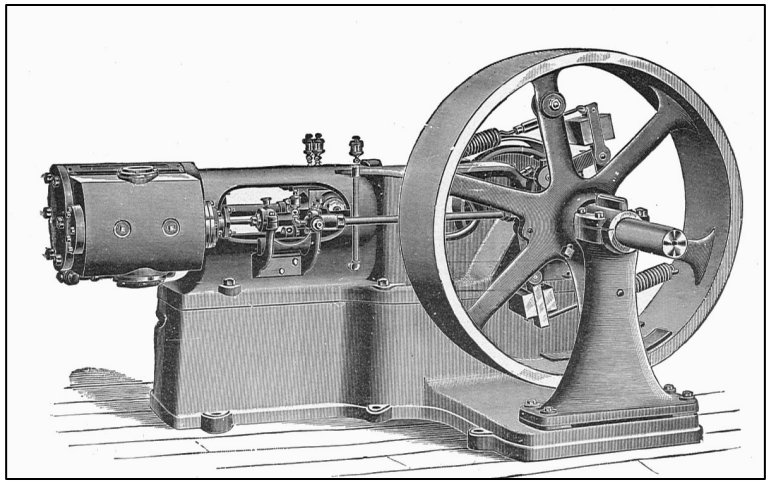
Analysis of programs

- A program is a physical object
  - some symbols on paper
  - a pattern of bits in memory
- But we have to analyze the meaning of the program
  - the dynamic system defined by the program
- This is program semantics.

ENTRA

Programs are machine that consume energy

```
n = 4;
z = 1;
while (n > 0) {
    z = z*n;
    n = n-1;
}
print(z);
```



Let us define the machine defined by a program.

ENTRA

Program semantics

```
n = 4;
z = 1;
while (n > 0) {
    z = z*n;
    n = n-1;
}
print(z);
```

To execute or analyse this program, we need to understand the meaning of "while", "semicolon", "{", "}", etc.

ENTRA

Mathematical machines

- Program semantics is based on transforming programs to automata.
  - timed automata
  - state machines
  - Petri nets, process formalisms
  - Kripke models
  - Turing machines

ENTRA

Syntax-directed semantics

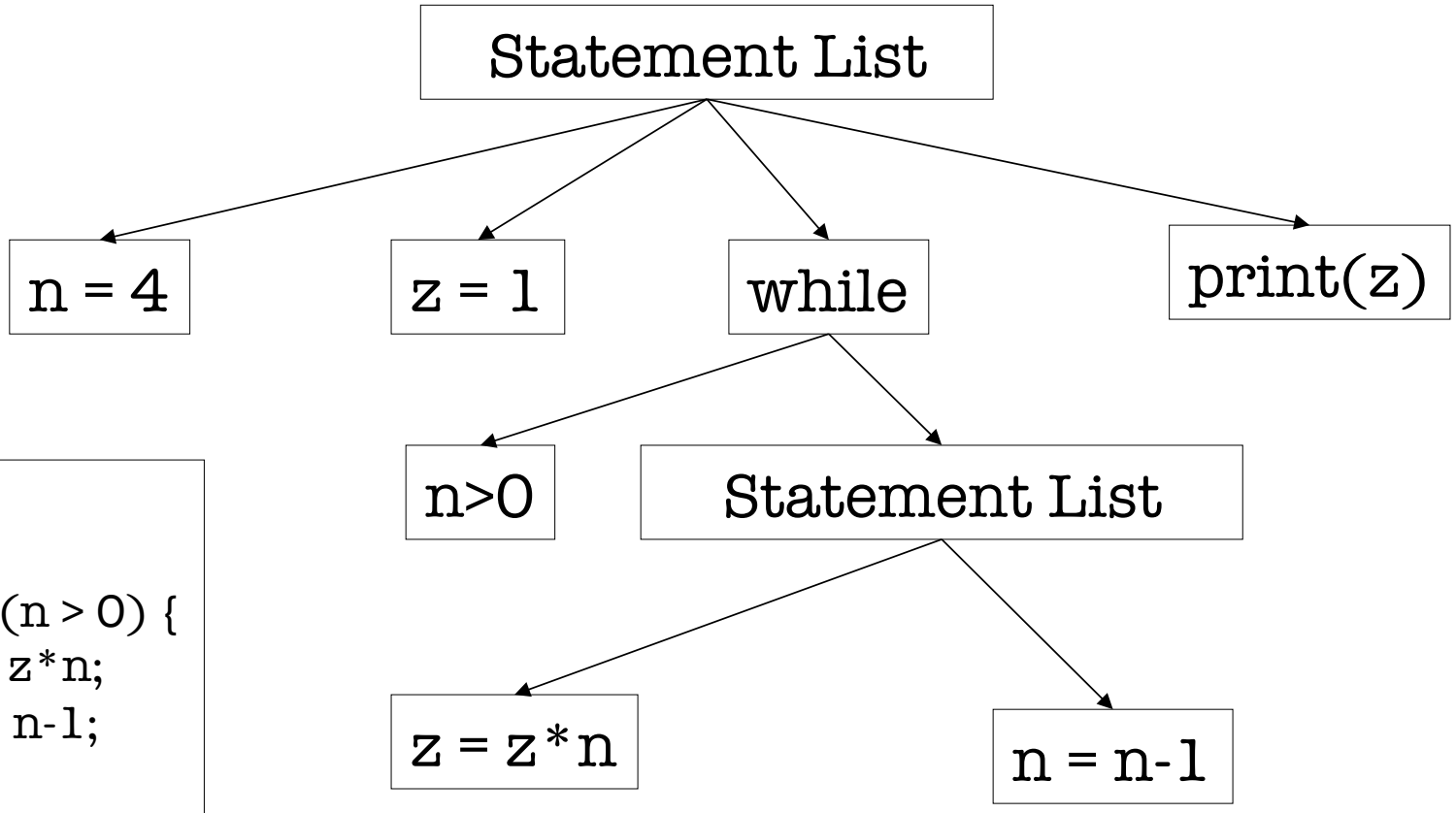
Big-step transitions follow the syntactic structure of the program. E.g.

$$\frac{\langle s_1, \sigma\rangle \Rightarrow \sigma' \quad \langle s_2, \sigma'\rangle \Rightarrow \sigma''}{\langle s_1 ; s_2, \sigma\rangle \Rightarrow \sigma''}$$

$$\frac{\langle s, \sigma\rangle \Rightarrow \sigma' \quad \langle \text{while } (b)\ s, \sigma'\rangle \Rightarrow \sigma''}{\langle \text{while } (b)\ s, \sigma\rangle \Rightarrow \sigma''} \qquad \text{if } b \text{ is true in } \sigma$$

$$\frac{}{\langle \text{while } (b)\ s, \sigma\rangle \Rightarrow \sigma} \qquad \text{if } b \text{ is false in } \sigma$$

Program syntax tree (parsing)




```
n = 4;
z = 1;
while (n > 0) {
    z = z*n;
    n = n-1;
}
print(z);
```


ENTRA

From syntax tree to flow graph

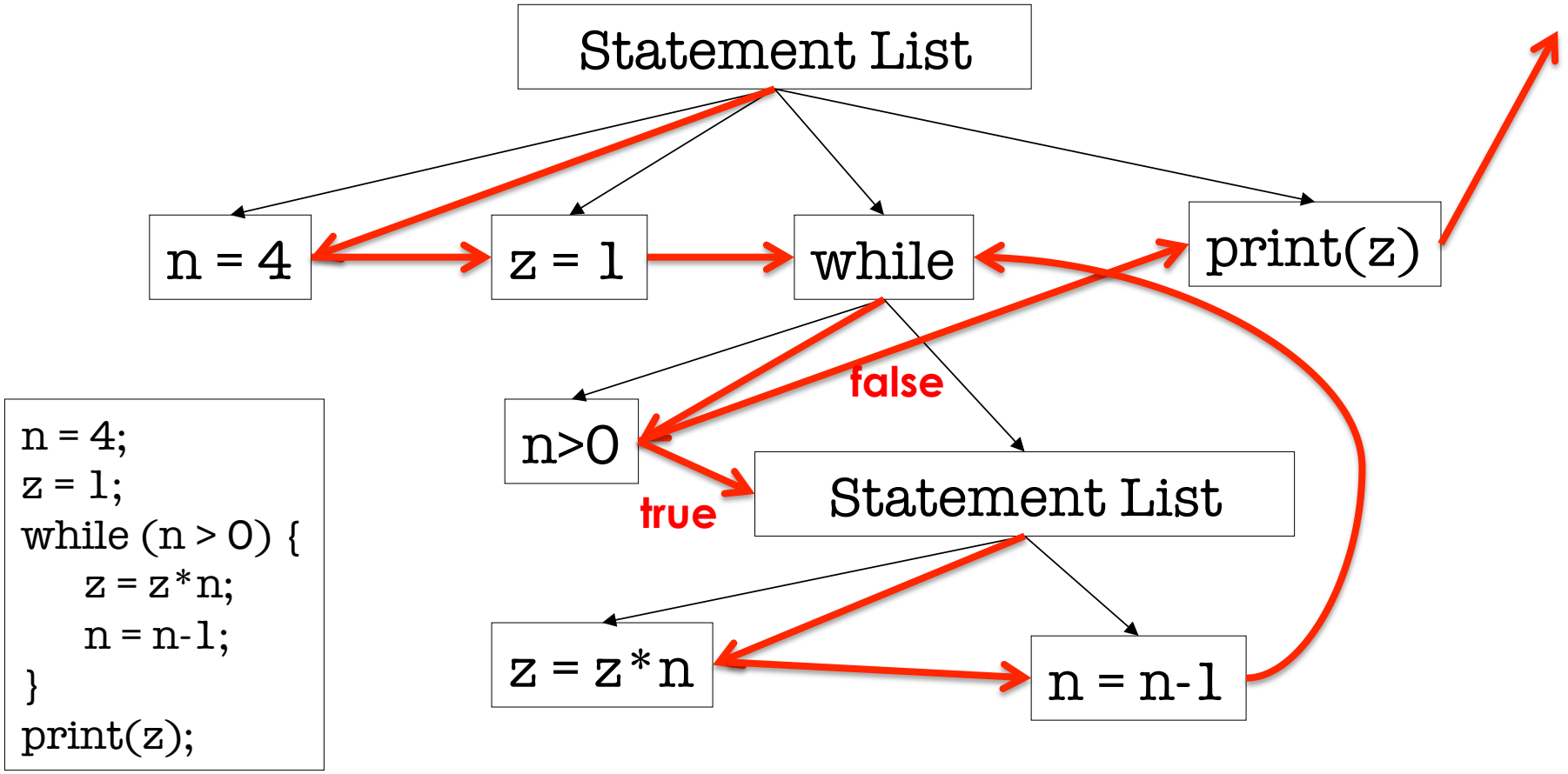
Grammar Rules	Semantic Rules for flow of control
If → if E then S_1 else S_2	E.true := S_1
	E.false := S_2
	S_1.next := If.next
	S_2.next := If.next
While → while E S_1	E.true := S_1
	E.false := While.next
	S_1.next := While
StatementList → $S_1 S_2$ ..... S_n	S_j.next = S_{j+1} (j = 1 to n-1)
	S_n.next := StatementList.next

S → StatementList | If | While | Print | Assign

StatementList.next := S.next
If.next := S.next
While.next := S.next
Print.next := S.next
Assign.next := S.next

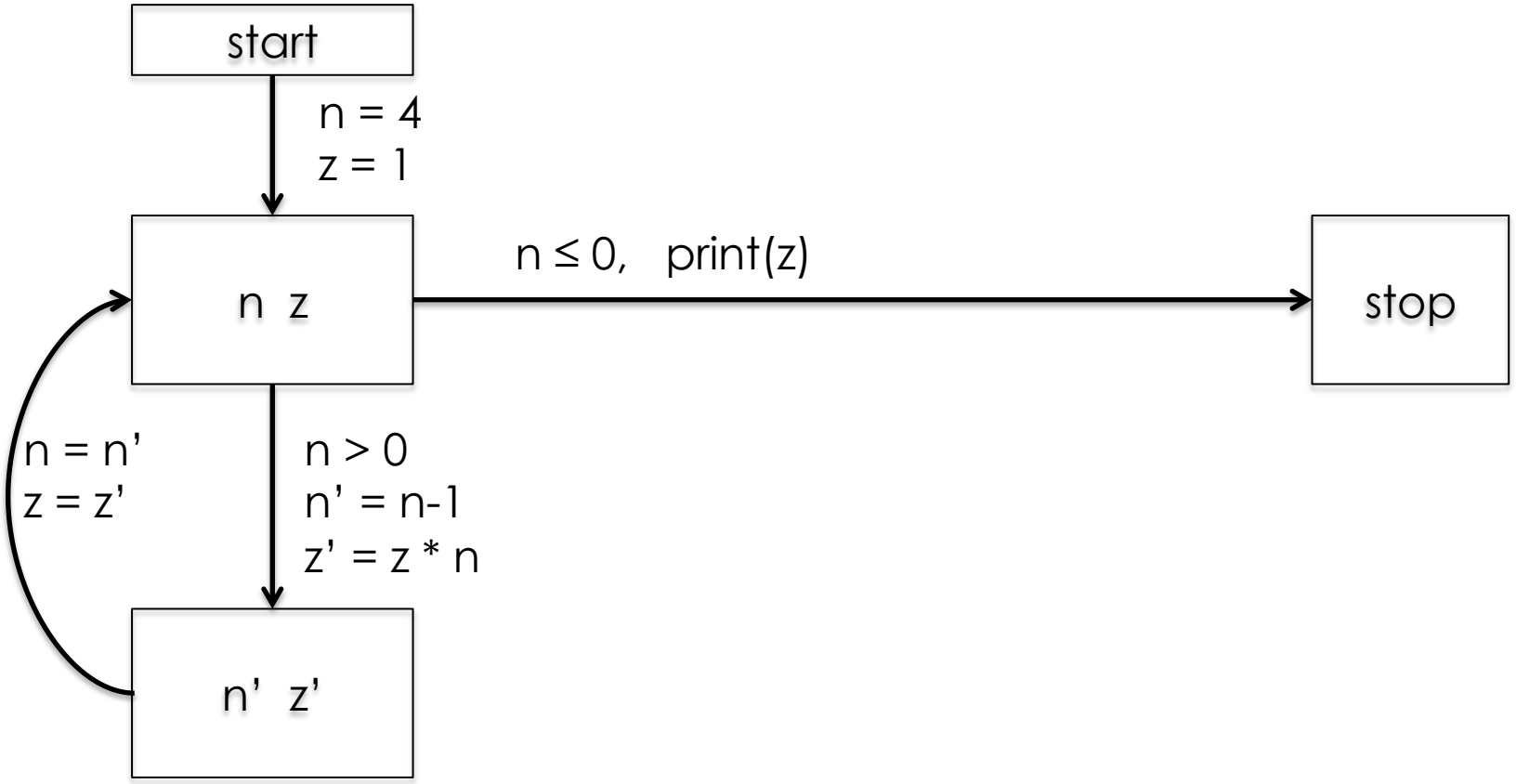
ENTRA

From syntax tree to flow graph




```
n = 4;
z = 1;
while (n > 0) {
    z = z*n;
    n = n-1;
}
print(z);
```

From flow graph to state automata



From automaton to predicate logic

$$\text{true} \rightarrow \text{reachable}_1$$

$$(\text{reachable}_1 \wedge n=4 \wedge z=1) \rightarrow \text{reachable}_2(n,z)$$

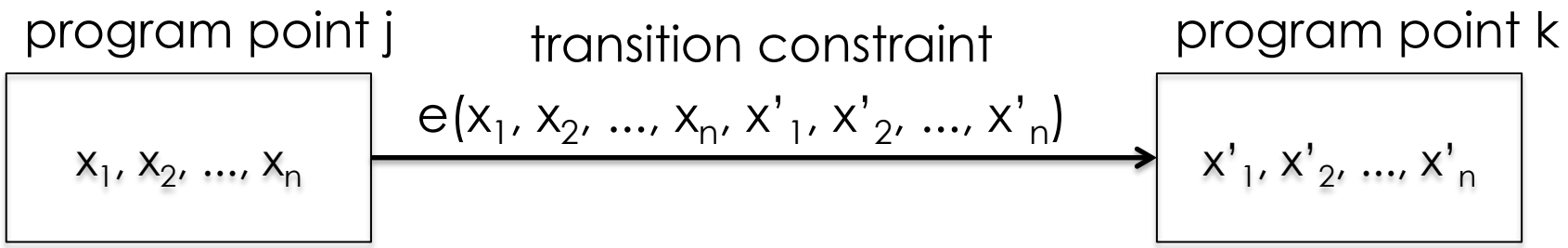
$$(\text{reachable}_2(n,z) \wedge n<0 \wedge z'=z*n \wedge n'=n-1) \rightarrow \text{reachable}_3(n',z')$$

$$(\text{reachable}_3(n',z') \wedge n=n' \wedge z=z') \rightarrow \text{reachable}_2(n,z)$$

$$\text{reachable}_2(n,z) \wedge n \geq 0 \wedge \text{print}(z)) \rightarrow \text{stop}$$

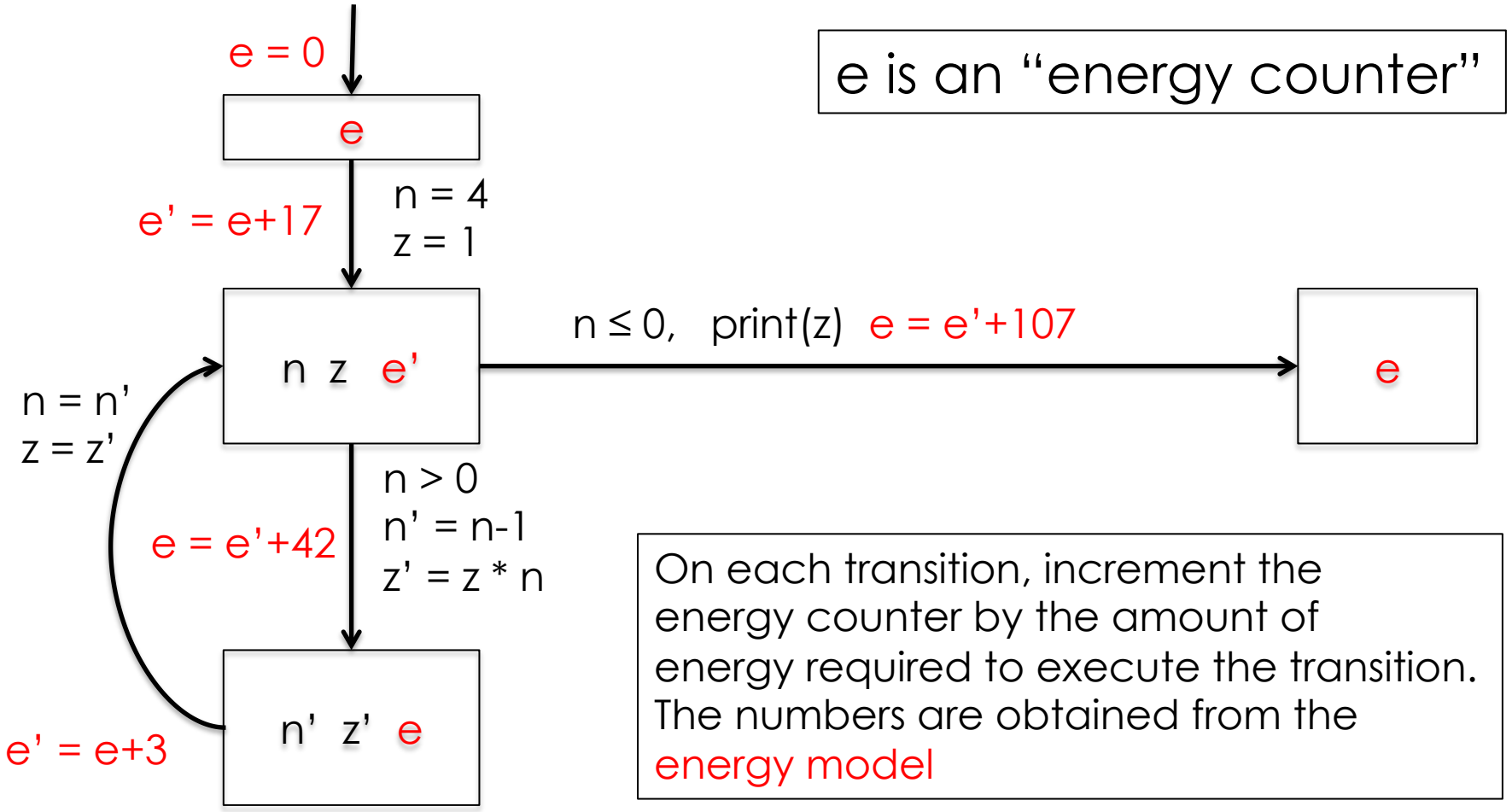
ENTRA

Logical representation



$$(\text{reachable}_j(x_1, x_2, ..., x_n) \wedge e(x_1, x_2, ..., x_n, x'_1, x'_2, ..., x'_n)) \rightarrow \text{reachable}_k(x'_1, x'_2, ..., x'_n)$$

Adding energy to the model



e is an "energy counter"

On each transition, increment the energy counter by the amount of energy required to execute the transition. The numbers are obtained from the energy model

ENTRA

Energy invariants

- The program state can contain resource counters.
- $\text{reachable}_k(x_1,\ldots,x_n,e)$ means that the total energy consumed is e, when the program reaches point k

- So we can express and prove assertions about energy (or other resources)

ENTRA

Analyzing energy consumption

- The total energy consumed by the program is given by the energy counter in the reachable "stop" state.
- For this example, the analysis yields a value of 304 (initial value n=4)
- However if the input data is unknown, we would get an invariant relationship between input value n and energy e.
- In the example, e = 17 + n*45 + 107

ENTRA

Deriving Invariants

- Many program analysis and verification tasks involve proving or deriving invariants

- An invariant is an assertion that is true at a given program point.

Proving invariants

- To prove that invariant P holds at program point j, prove the following implication

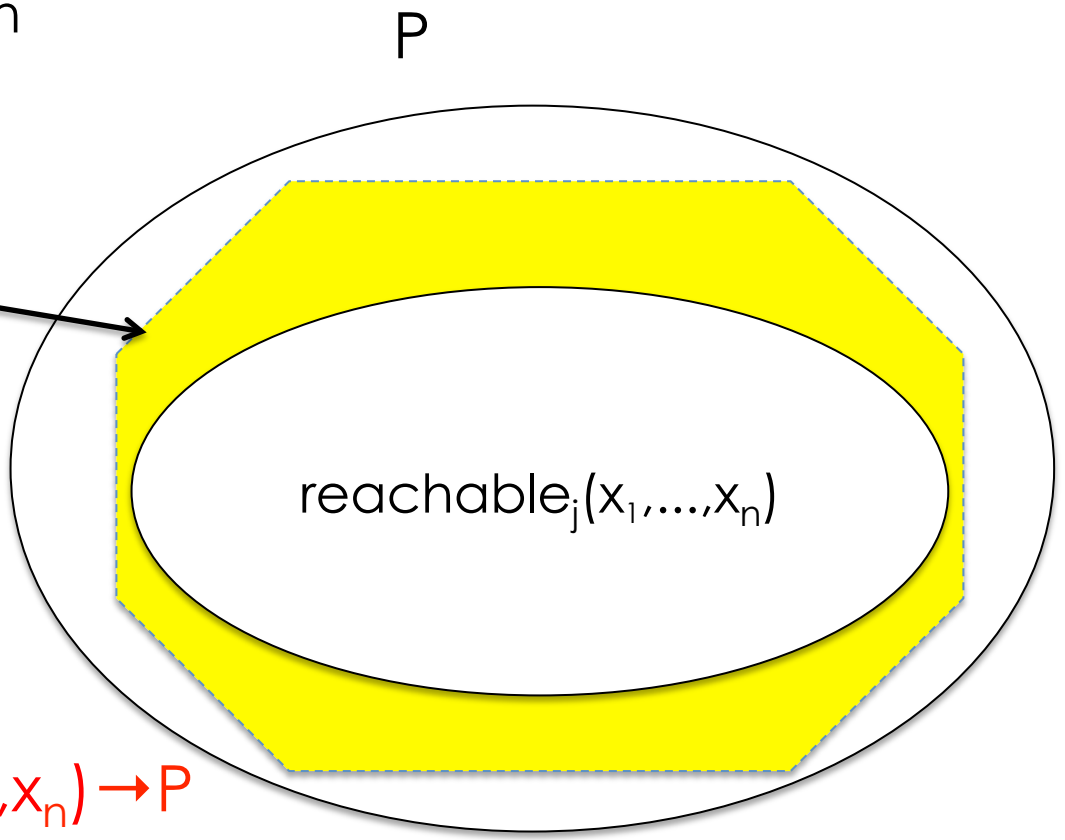
$reachable_j(x_1,\ldots,x_n) \rightarrow P$

which is equivalent to

$\neg(reachable_j(x_1,\ldots,x_n) \wedge \neg P)$

Proof by approximation

Overapproximation of the set of points where $reachable_j(x_1,\ldots,x_n)$ is true.



Contained within P, hence

$reachable_j(x_1,\ldots,x_n) \rightarrow P$

ENTRA

Two basic techniques

- How to capture all reachable states?
  - answer, fixpoint techniques
- How to handle an infinite set of states?
  - answer, approximate using abstract interpretation
- These two methods underlie much program analysis

ENTRA

Example convex polyhedron abstraction

```
var i,j:int;
begin
  i=0; j=10;
  while i<=j do
    i = i+2;
    j = j-1;
  done;
end
```

```
r1(I,J) :-
      I=0,J=10.
r2(I,J) :-
      r1(I,J).
r2(I,J) :-
      I1 =< J1,
      I = I1+2,
      J = J1-1,
      r2(I1,J1).
r3(I,J) :-
      I >= J+1,
      r2(I,J).
```


ENTRA

Approximate reachable states

```
r1(I,J) = [I=0,J=10].
r2(I,J) = [-I >= -16,I >= 0,I+2*J=20].
r3(I,J) = [-3*I >= -26,3*I >= 22,I+2*J=20].
```

This result is computed fast, using the Parma Polyhedra Library to perform the operations on convex polyhedra.

ENTRA

Example

```
13  int biquadCascade(biquadState &state, int xn) {
14      unsigned int ynl;
15      int ynh;
16
17      for(int j=0; j<BANKS; j++) {
18          ynl = (1<<(FRACTIONALBITS-1));
19          ynh = 0;
20          {ynh, ynl} = macs( biquads[j].b0, xn, ynh, ynl);
21          {ynh, ynl} = macs( biquads[j].b1, state.b[j].xn1, ynh, ynl);
22          {ynh, ynl} = macs( biquads[j].b2, state.b[j].xn2, ynh, ynl);
23          {ynh, ynl} = macs( biquads[j].a1, state.b[j+1].xn1, ynh, ynl);
24          {ynh, ynl} = macs( biquads[j].a2, state.b[j+1].xn2, ynh, ynl);
25          if (sext(ynh,FRACTIONALBITS) == ynh) {
26              ynh = (ynh << (32-FRACTIONALBITS)) | (ynl >> FRACTIONALBITS);
27          } else if (ynh < 0) {
28              ynh = 0x80000000;
29          } else {
30              ynh = 0x7fffffff;
31          }
32          state.b[j].xn2 = state.b[j].xn1;
33          state.b[j].xn1 = xn;
34
35          xn = ynh;
36      }
37      state.b[BANKS].xn2 = state.b[BANKS].xn1;
38      state.b[BANKS].xn1 = ynh;
39      return xn;
```

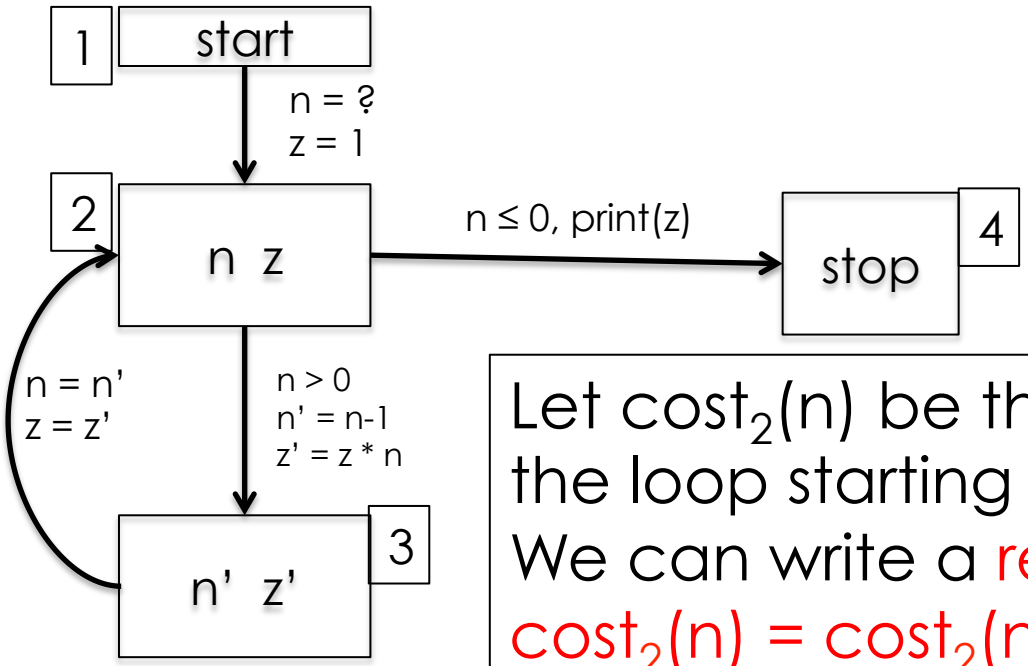
biquadCascade(BANKS)
=
157 * BANKS + 51.7
nJoules

This is an estimate of the energy used by the function.

It is a linear function of the value of BANKS

ENTRA

Deriving cost functions



Let $cost_2(n)$ be the cost of the loop starting at 2.
We can write a recurrence relation
$cost_2(n) = cost_2(n-1) + 45$ (if $n > 0$)
$cost_2(n) = 0$ (if $n \leq 0$)
The cost of the whole computation for input n is $17 + cost_2(n) + 107$

Solving cost relations

- Tools like Mathematica are capable of solving many recurrence relations.

$cost_2(n) = cost_2(n-1) + 45$ (if $n > 0$)

$cost_2(n) = 0$ (if $n \leq 0$)

has a closed-form solution

$cost_2(n) = 45*n$

More complex cases

- By solving energy recurrence equations we can get non-linear energy functions
- E.g. a matrix multiplication program for matrices of size n

  $42.47\ n^3 + 68.85\ n^2 + 49.9\ n + 24.22$ nJoules

ENTRA

Energy and multi-threaded code

- Often, we want to design threads to run as slowly as possible, while still meeting performance targets
- Reducing clock frequency saves power
- Cores that are inactive should be put in power-saving modes

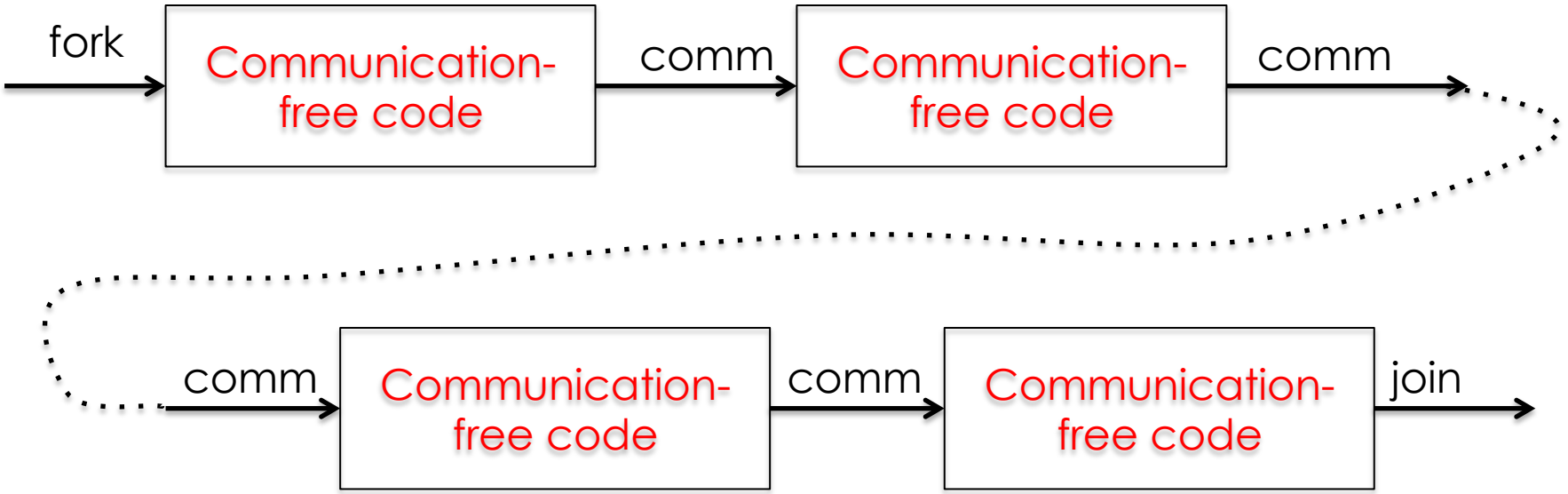
ENTRA

Communication and timing

- We consider a language with synchronous channel communication
- The programmer needs an analysis of how much work and time a thread uses between communications

ENTRA

Behaviour of a single thread



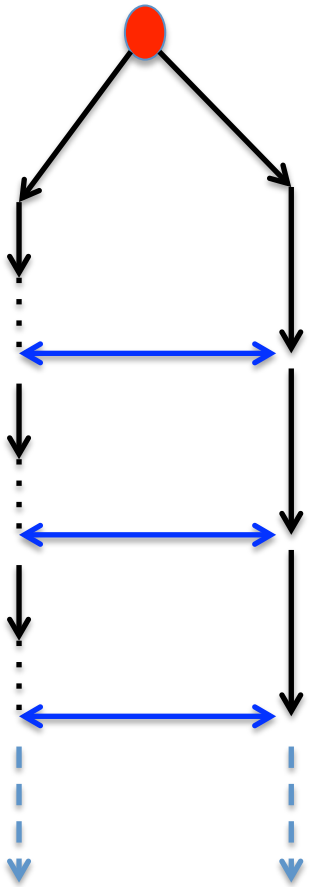
Parallel execution

Timing analysis is vital.

E.g. suppose the left thread always waits for the other.

Possible energy optimisations:

1. slow down the left thread
2. give it some more work to balance the load
3. put in power-saving mode while waiting

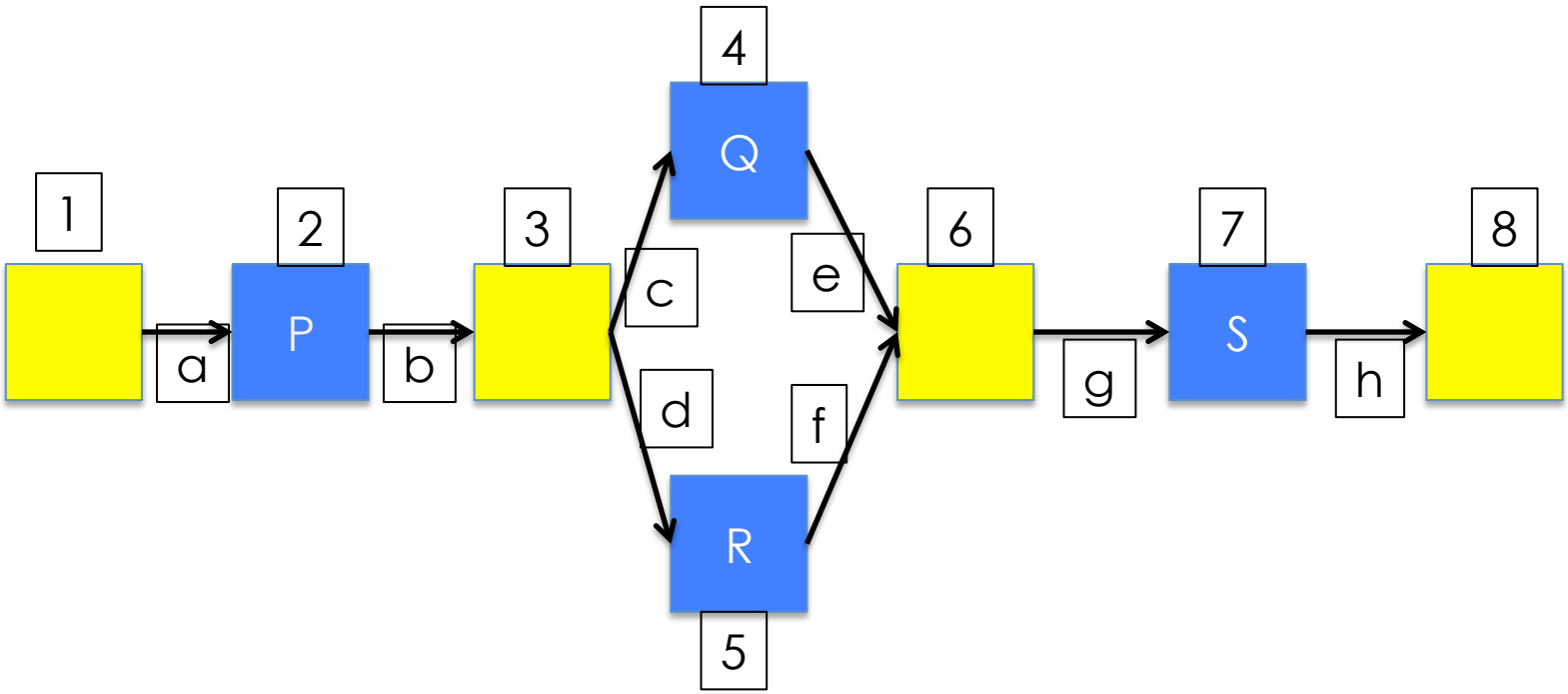


The threads run until they reach a synchronisation point.

After synchronising, they continue to the next, etc.

ENTRA

Example thread behaviour

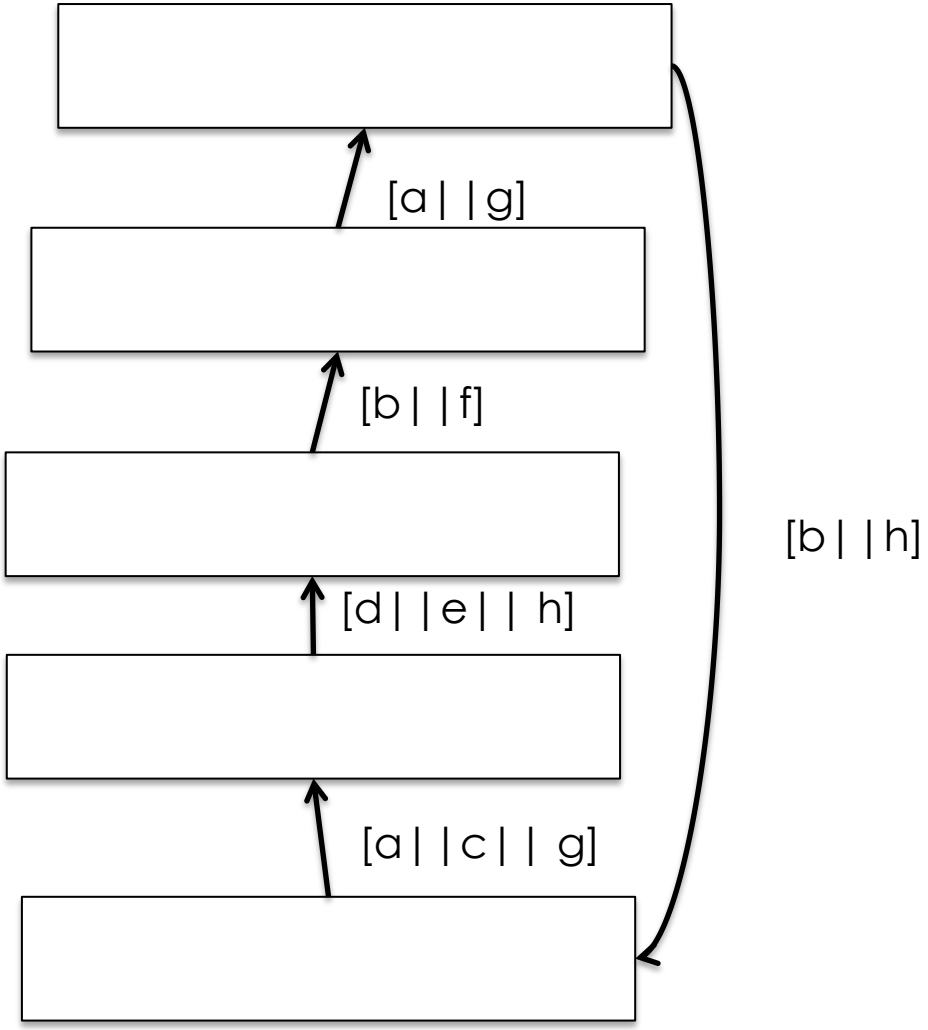


8 threads in a pipeline with a split in the middle.
P,Q,R and S are some functions on the values passed along.

ENTRA

Analysis of the thread synchronisation identifies periodic behaviour

Compute a critical path of the loop, based on the time estimates and the order on tasks.



Thread behaviour

- Assume task times
  - P = 100, Q = 334, R=500, S=250
- Obtain throughput
  - 255
- Thread activity
  - Thread 7 (67%), Thread 5 (66%), Thread 4 (44%),..... Thread 1(1.3%)

ENTRA

Energy and power estimates

- The energy of the whole cycle consists of
  - the energy for each task in the cycle
  - an overhead for the number of active threads (obtained from the critical path)
  - an estimate of the energy used while idling
- The power (Watts) is E/T, where E is the energy and T is the time of the cycle

ENTRA

Energy a design goal for programmers

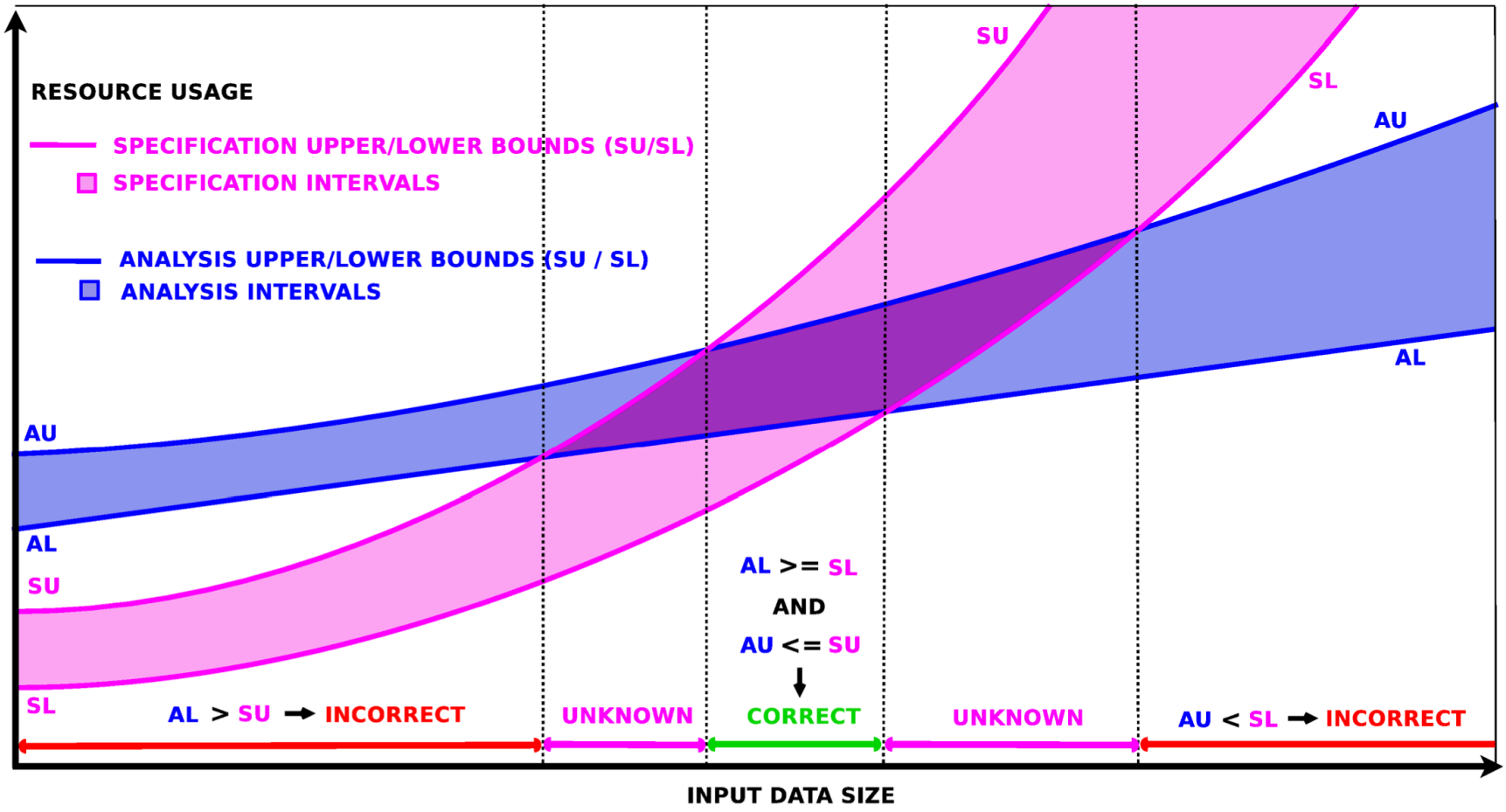
```
#pragma check energy(proc(x))<5pJ
int proc(int x) {
...
```

```
Output:

Checked 0≤x≤5⇒energy(proc(x))<5pJ
```


ENTRA

Verification of energy specifications



ENTRA

What can energy transparency achieve?

- By analysis combined with an energy model, we can
  - estimate energy cost for the whole program (parameterised by input data, and hardware settings in the energy model)
  - estimate the relative energy costs of program parts, find hot spots
  - estimate power dissipation when combined with timing analysis
  - suggest areas for energy optimization

ENTRA

Limitations on energy transparency

- The energy model could be too complex to be practical
- E.g. consider energy modelling of cloud computing
  - energy of data transport
  - energy of computation
  - energy of cooling
  - energy of client machine
- It is an open question whether these can be modelled accurately enough to optimise cloud-based applications

ENTRA

Limitations (2)

- The analysis might be complex
- Complex software is extremely difficult to analyze
  - threads
  - non-determinism due to caches, pipelines, races
  - operating system effects
- Rough approximations of system behaviour are used

ENTRA

Thank you

ENTRA