



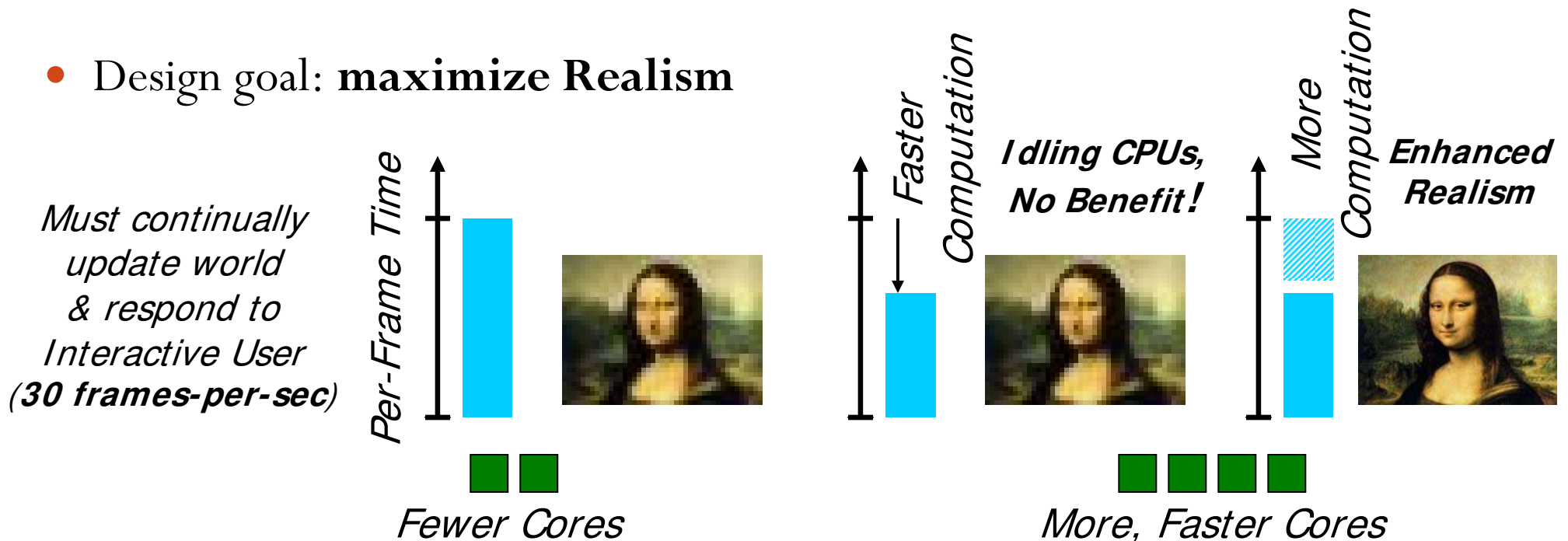
Opportunistic Computing: A New Paradigm for Scalable Realism on Many-Cores

Romain Cledat, Tushar Kumar,
Jaswanth Sreeram, and **Santosh Pande**



Speedup Is Not Always the End-Goal

- **Immersive Applications** intend to provide the *richest, most engrossing experience* possible to the *interactive* user
 - Gaming, Multimedia, Interactive Visualization
- With growing number of cores, or increasing clock-frequencies
 - These applications want to do *MORE*, not just do it *FASTER*
- Design goal: **maximize Realism**



What is Realism?

- Realism consists of
 - **Sophistication in Modeling**
 - Example: Render/Animate as highly detailed a simulated world as possible
 - **Responsiveness**
 - Example: Update world frequently, respond “instantly” to user inputs
 - Unit of world update: **Frame**
- Typical Programming Goal
 - Pick models/algorithms of as high a sophistication as possible that can execute within a *frame deadline* of $1/30$ seconds
- Flexibility: *Probabilistic Achievement of Realism is Sufficient*
 - Most frames (say, $>90\%$) must complete within 10% of frame deadline
 - Relatively few frames ($<10\%$) may complete very early or very late

How do we Maximize Realism?

Maximizing Realism

*Two
complementary
techniques*

#1: N-version Parallelism

Speed up hard-to-parallelize algorithms with high probability using more cores

- Applies to algorithms that make random choices
- Basic Intuition: Randomized Algorithms (but not limited to them)

#2: Scalable Soft Real-Time Semantics (*SRT*)

Scale application semantics to available compute resources

- Applies to algorithms whose execution time, multi-core resource requirements and sophistication are parametric
- Basic Intuition: Real-Time Systems (but with different formal techniques)

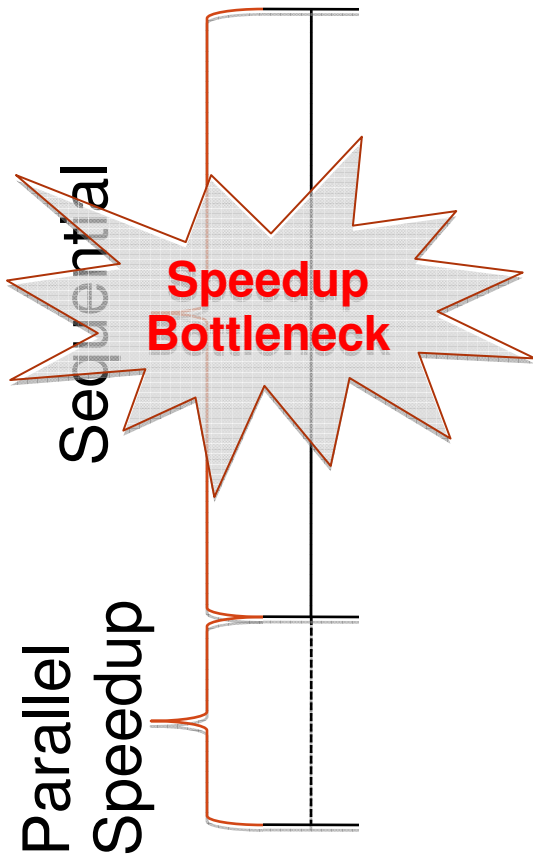
Unified as *Opportunistic Computing Paradigm*:
N-versions creates slack for **SRT** to utilize for Realism

#1

N-Versions Parallelism:

Speedup Sequential Algorithms with
High Probability

Bottleneck for Speedup



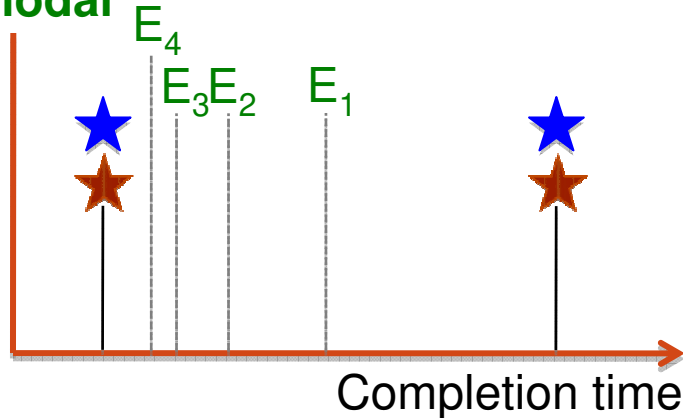
- Applications still have significant sequential parts
 - Stagnation in processor clock frequencies makes sequential parts the major bottleneck to speedup (Amdahl's Law)
- A reduction in *expected execution time* for sequential parts of an application will provide more slack to improve realism

Intuition

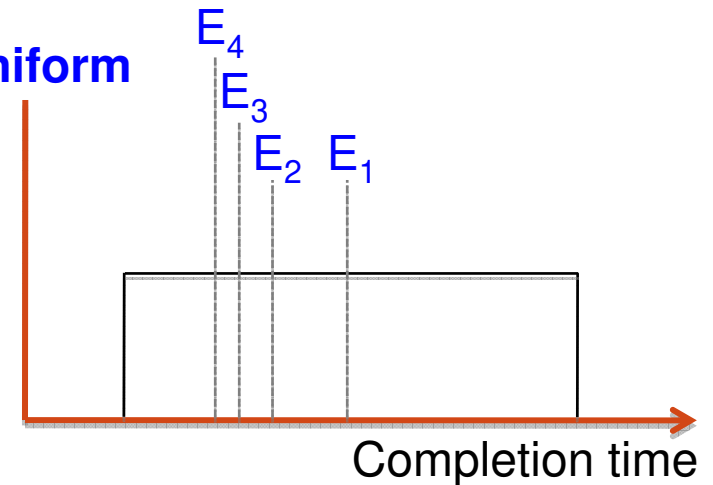
- Algorithms making random choices for a fixed input lead to varying completion times

Run 2 instances in parallel under isolation

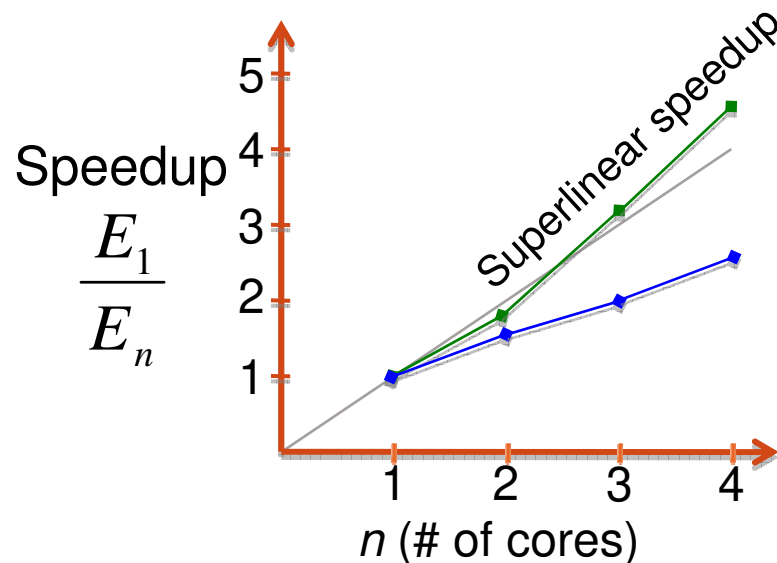
Bimodal



Uniform

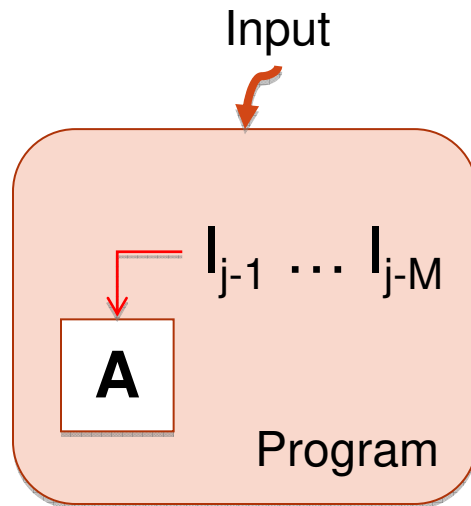


Fastest among 2 is faster than average with high probability

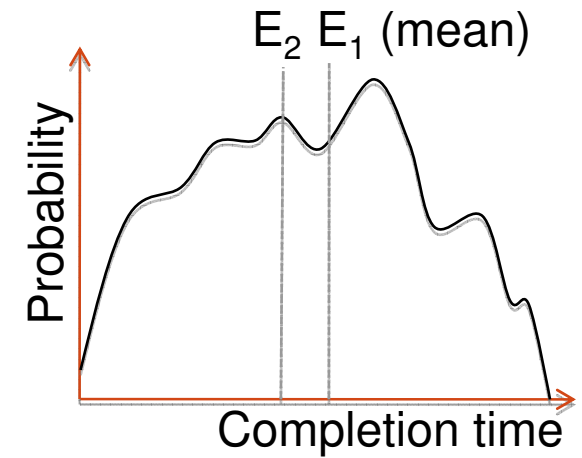


- Big opportunities for expected speedup with increasing n
- Tradeoff $S = \frac{E_1}{E_n} \leftrightarrow n$
 - Requires knowledge of distribution
 - Wider spread \rightarrow more speedup

Application Use Scenario



- **Goal:** Find the reasonable n to reduce expected completion time of $PDF[A(I_j)]$



- **Need** knowledge of $PDF[A(I_j)]$ to compute the speedup S
- **Determine** $PDF[A(I_{j-1}) \dots A(I_{j-M})]$ How do we do this?
- **Assume** $PDF[A(I_j)] \approx PDF[A(I_{j-1}) \dots A(I_{j-M})]$ (stability condition) When will this hold?
 - Stability condition gives predictive power

We want to determine the speedup S and the number of concurrent instances n on $A(I_j)$ from PDF with *no prior* knowledge of the underlying distribution

PDF and Stability Condition

$$PDF[A(I_j)] \approx PDF[A(I_{j-1}) \dots A(I_{j-M})]$$

- Holds **statically** over j for inputs of the same “size”
 - Graph algos: $|V|$ and $|E|$
- Holds for **sufficiently slow variations**
 - $|I_{j-M}| \approx \dots \approx |I_{j-1}| \approx |I_j|$
- Example: TSP for trucks in continental United States
 - Fixed grid size
 - Similar paths

- Randomized algorithms
 - Analytically known PDF
 - Depends on input *size* and *parameters* (referred to as “size”)
 - “Size” might be unknown
- Other algorithms
 - PDF is analytically unknown/intractable



Runtime Estimation

N-version parallelism in C/C++

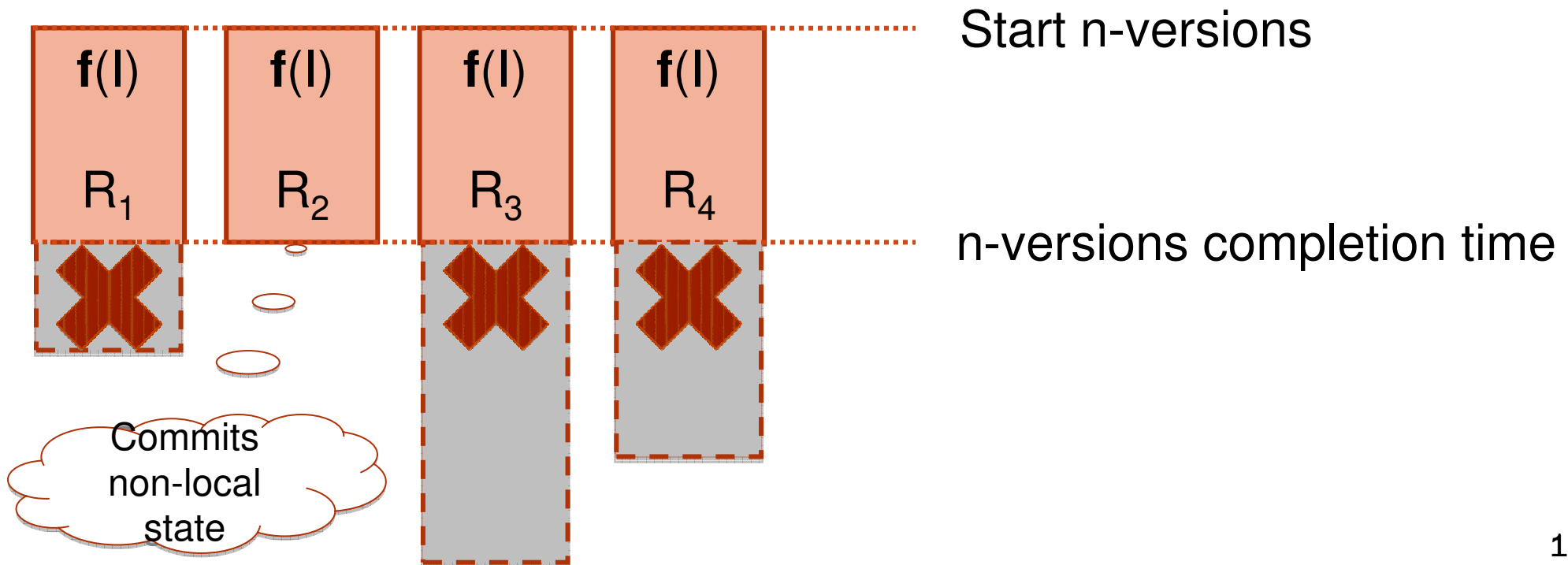
```
int a[];
void f(Input) {
    int b = ...;
    a[k] = ...;
}
```

C++ can eliminate API wrappers
Shared<int> a[];

Local state: leave as is

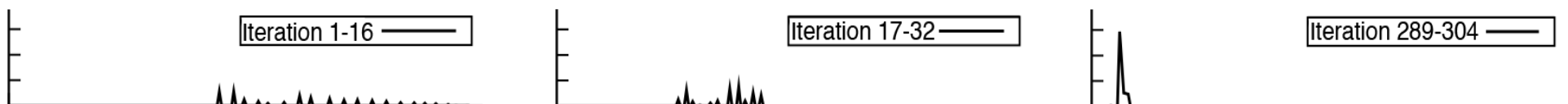
Non-local state: wrap with API call

Render each instance side-effect free



Current Avenues of Research

- How **broad** is the class of algorithms that
 - Make random choices
 - Satisfy the stability condition
- Exploring common randomized algorithms
 - TSP over a fixed grid
 - Randomized graph algorithms
- Exploring applicability of our technique to application specific characteristics that indirectly benefit performance
 - Reducing the *number of iterations* in a Genetic Algorithm by minimizing the *expected score* at each iteration
- Or, achieving a better *final score* (higher **quality of result**)
 - Independent of performance gains



#2

Scalable Soft Real-Time Semantics (SRT):
Scale Application Semantics to
Available Compute Resources

Applications with Scalable Semantics

- Games, Multimedia Codecs, Interactive Visualization
 - Possess scalable semantics



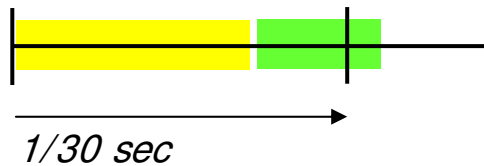
*Game-Frames
at approx. 30 fps*

Characteristic 1

User-Responsiveness is Crucial.

➔ Model/Algorithmic Complexity must be suitably adjusted / bounded

Frame Time



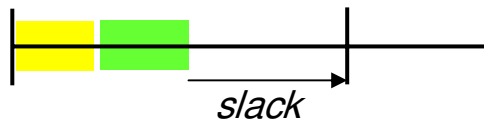
Frame# 0 - 10

*Scale down AI complexity:
think-frequency, vision-range*

Characteristic 2

**Dynamic Variations
in Execution Time over Data Set.**

➔ To preserve Responsiveness while maximizing Sophistication,
Continually Monitor Time and Scale
Algorithmic Complexity (semantics)

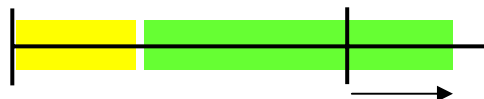


*compromises Realism
by not maximizing
Sophistication*



Frame# 50 - 60

*Scale up AI & Physics complexity:
sim time-step, effects modeled*



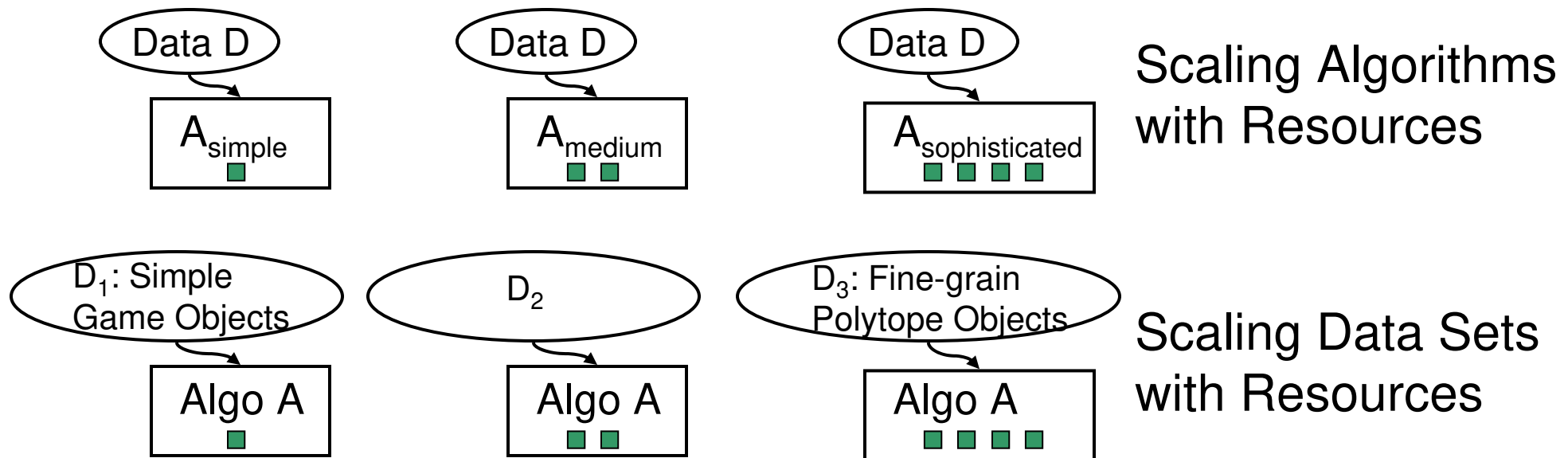
Frame# 80 - 90

Scale down Physics complexity

*Missed deadline significantly
Responsiveness Affected*

Scaling Semantics with Multi-cores

- Traditionally, benefiting from more cores required breaking up the same computation into more parallel parts
 - Difficult problem for many applications, including gaming and multimedia
- Scalable Semantics provide an *additional* mechanism to utilize more cores



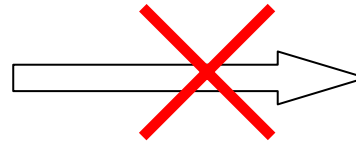
***Scripted Game-World Interactions,
Unbreakable Objects***



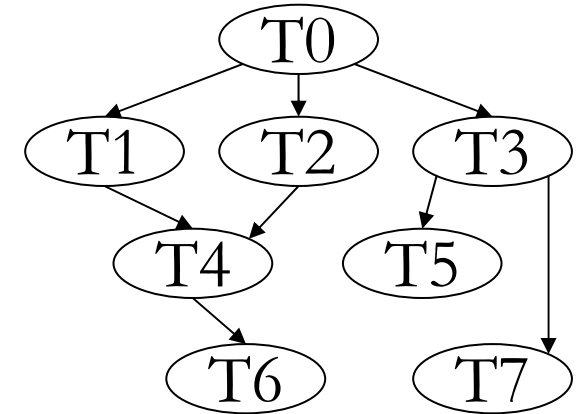
***Open-Ended Game-World Interactions,
Dynamic Fracture Mechanics***

Don't Real-Time Methods Solve This Already?

Games, Multimedia, Interactive Viz



*Implement as
a Real-Time App*



Real-Time Task-Graph

- Application decomposed into Tasks and Precedence Constraints
- Responsiveness guaranteed by Real-time semantics (hard or probabilistic)

*Implement with
High-Productivity,
Large Scale
Programming flows*

C, C++ , Java: Monolithic App

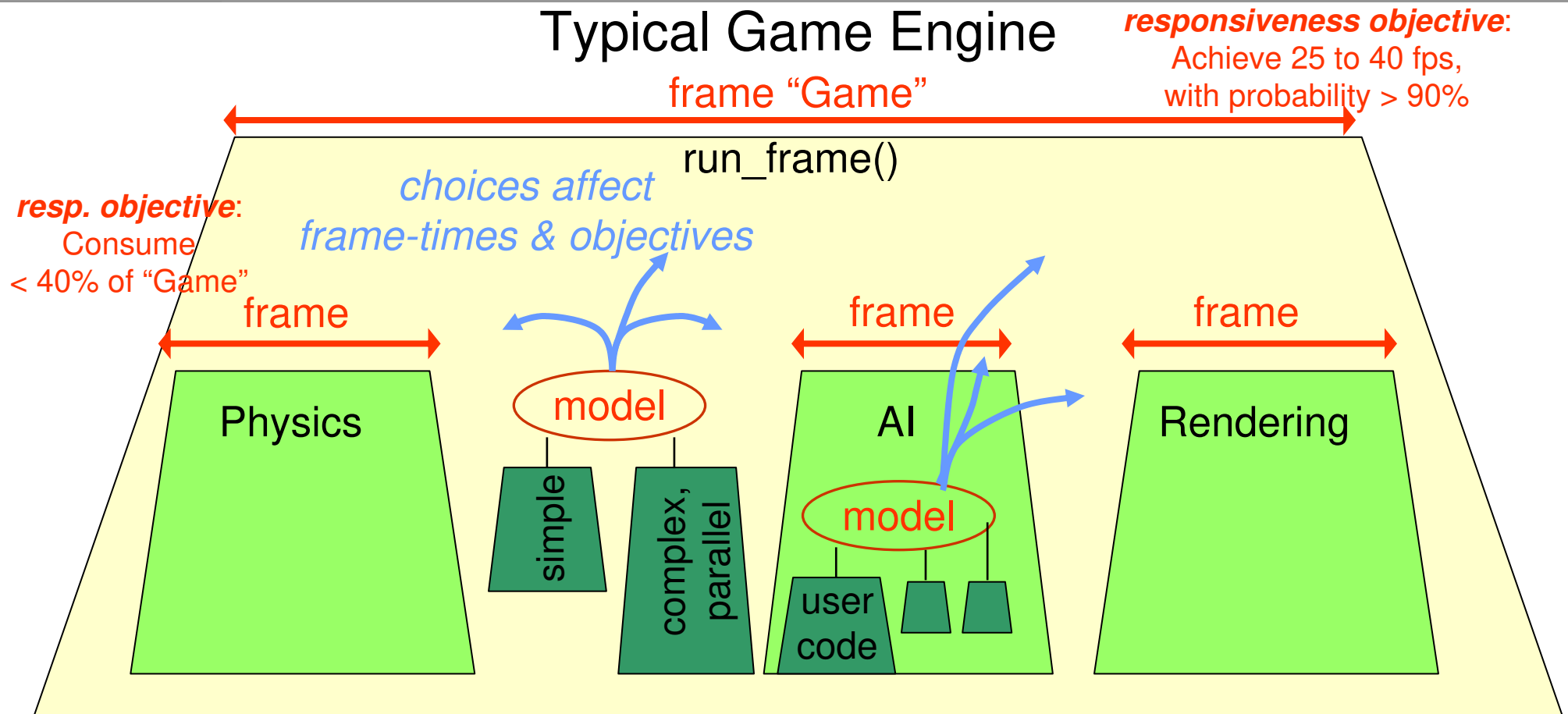
- 100Ks to Millions of LoC
- No analyzable structure for responsiveness and scaling
- Responsiveness is entirely an **emergent** attribute
(currently tuning this is an art)

**Need a new bag of tricks to Scale
Semantics in Monolithic Applications**



Scaling Semantics in Monolithic Applications

- Challenge for Monolithic Applications
 - C/C++/Java do not express user-responsiveness objectives and scalable semantics
- Our Approach
 - Let **Programmers** specify *responsiveness policy* and *scaling hooks* using SRT API
 - Let **SRT Runtime** determine *how* to achieve policy by manipulating provided hooks
- SRT API enables programmers to specify policy and hooks
 - Based purely on their knowledge of the **functional design** of individual algorithms and application components
 - Without requiring them to anticipate the **emergent responsiveness behavior** of interacting components
- SRT Runtime is based on **Machine Learning** and **System Identification** (Control Theory), enabling Runtime to
 - *Infer* the structure of the application
 - Learn *cause-effect* relationships across application structure
 - *Statistically predicts* how manipulating hooks will scale semantics in a manner that best achieves desired responsiveness policy

Case Study: Incorporating SRT API & Runtime in a Gaming Application



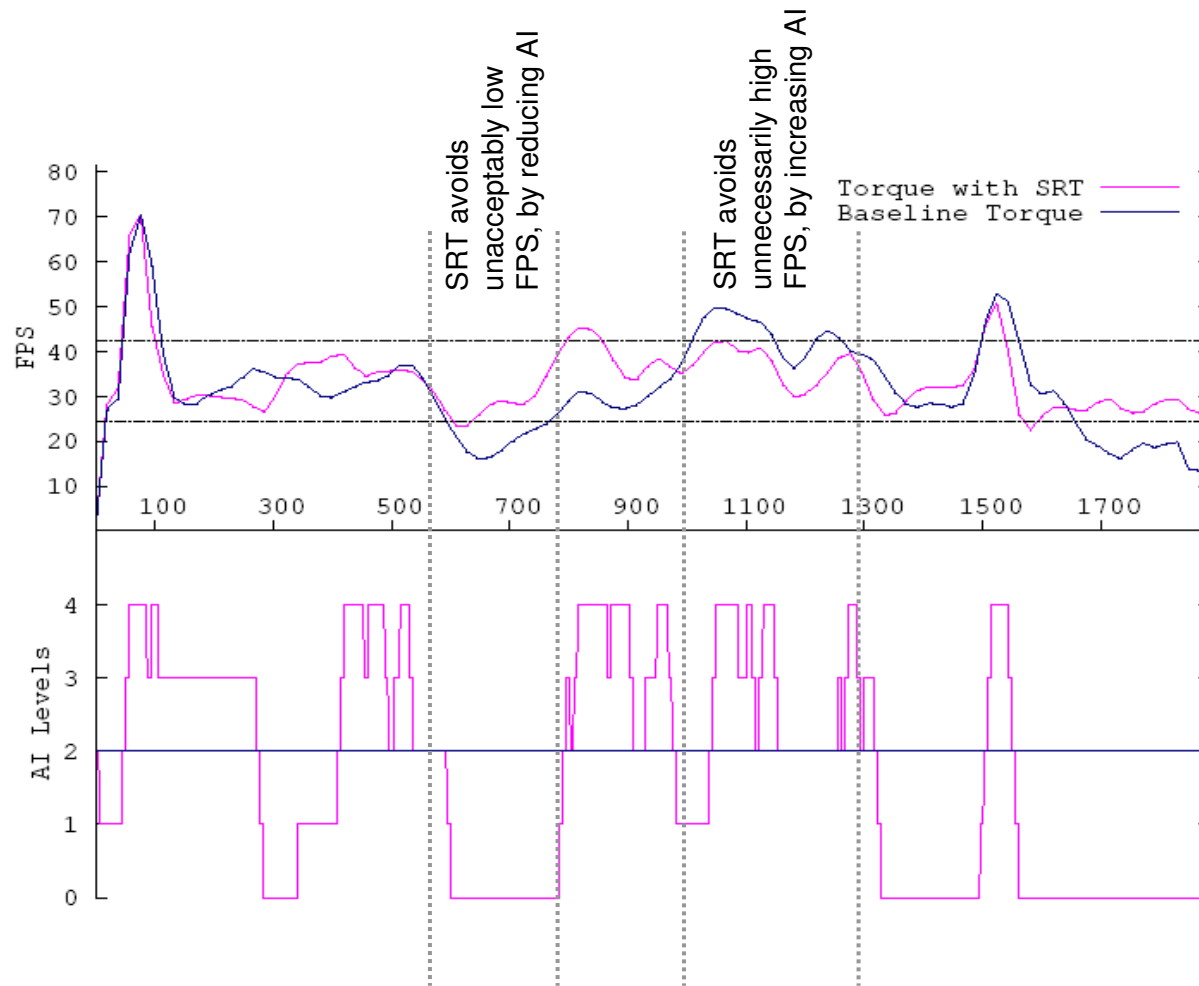
SRT Runtime

- Monitors **frame**
- Learns Application-wide **Average Frame Structure**
- Chooses between   user-codes in **model**

- Learns & Caches statistical relations:

- **Reinforcement Learning:** Which **models** predominantly affect which **objectives**? (*infer complex relationships, slowly*)
- **Feedback Control:** Adjust choices in **models** (**simple**, **medium**, **complex**, ...) to meet **objectives** (*fast reaction*)

Torque Game Engine: Measured Behavior



objective:
25 to 42 fps

Conclusion

- Maximizing Realism is underlying design goal for an important class of applications
 - Speedup is only one enabling factor
- Realism provides avenues to utilize multi/many-cores, over and above traditional task and data parallelism techniques
- We introduced two complementary techniques that utilize extra cores for maximizing Realism
 - **N-versions Parallelism:** Creates slack on hard to parallelize code
 - **Semantics Scaling SRT:** Utilizes dynamically available slack to maximize realism

Thank you!

- Questions?