Discrete Time and Race Conditions

Sean Williams, PhD

http://rio-lang.org

Discreteness

Race conditions are where order is important but ambiguous

This is a modeling phenomenon

Discreteness

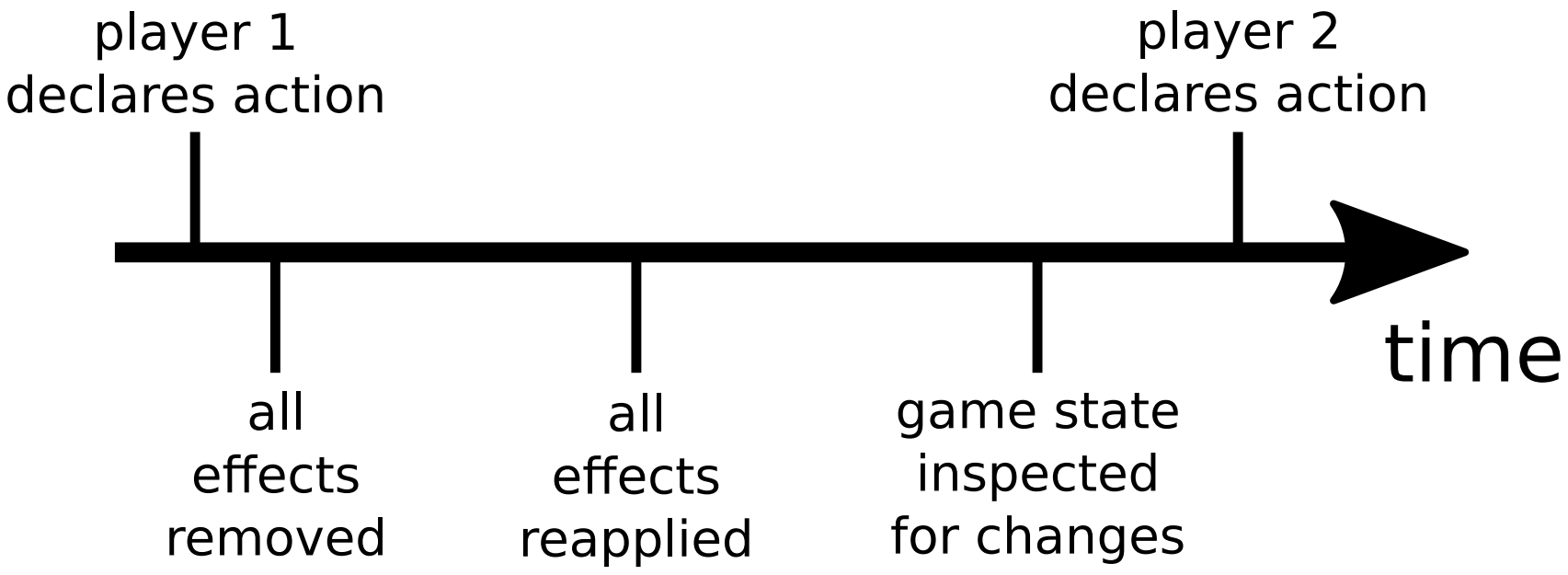
Types

Abilities

Power/Toughness

Discreteness

Discreteness



Discreteness

613. Interaction of Continuous Effects

613.1. The values of an object's characteristics are determined by starting with the actual object. For a card, that means the values of the characteristics printed on that card. For a token or a copy of a spell or card, that means the values of the characteristics defined by the effect that created it. Then all applicable continuous effects are applied in a series of layers in the following order:

613.1a *Layer 1*: Copy effects are applied. See rule 706, "Copying Objects."

613.1b *Layer 2:* Control-changing effects are applied.

613.1c *Layer 3:* Text-changing effects are applied. See rule 612, "Text-Changing Effects."

613.1d *Layer 4:* Type-changing effects are applied. These include effects that change an object's card type, subtype, and/or supertype.

613.1e *Layer 5:* Color-changing effects are applied.

613.1f *Layer 6:* Ability-adding effects, ability-removing effects, and effects that say an object can't have an ability are applied.

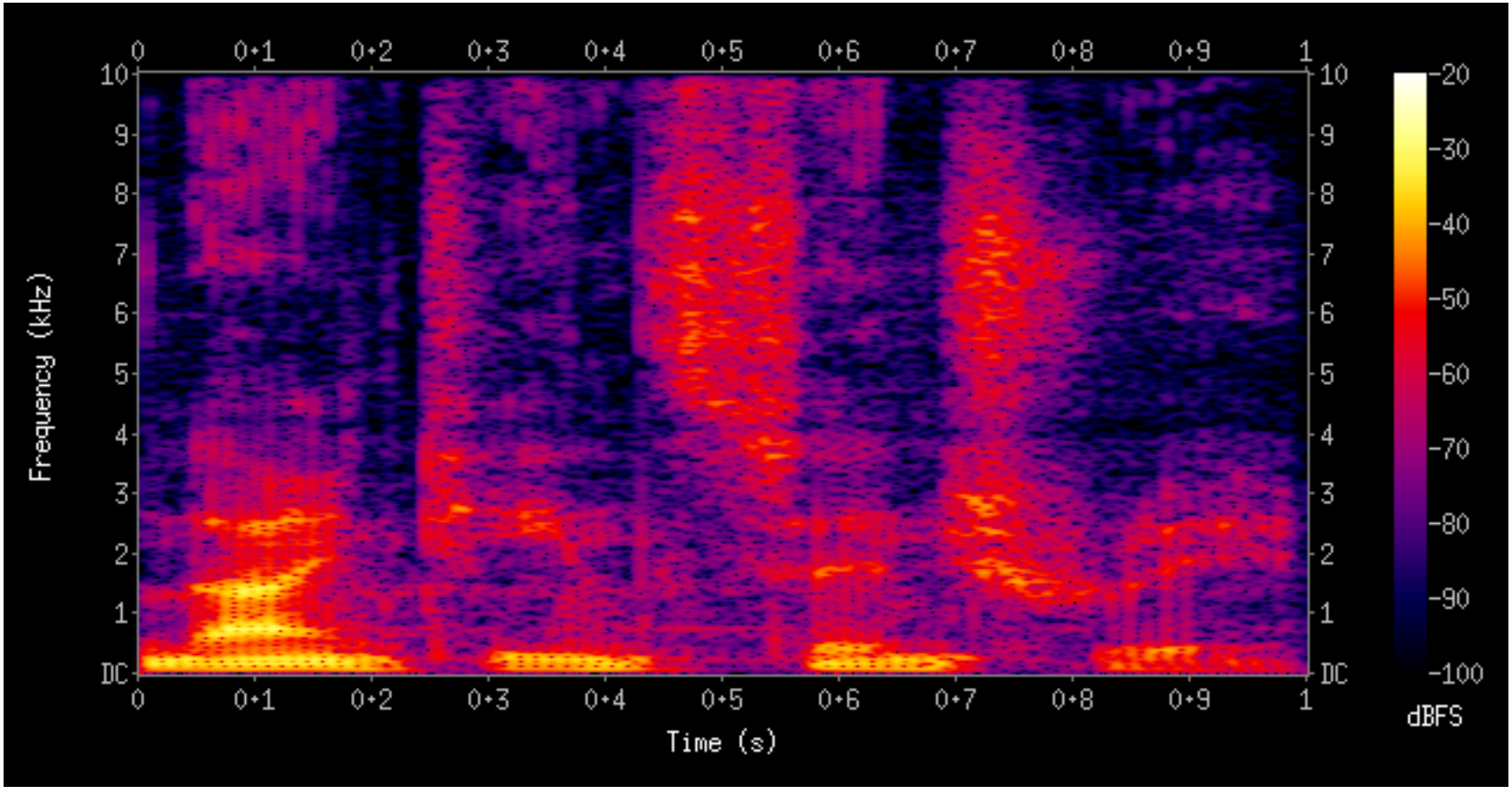
613.1g *Layer 7:* Power- and/or toughness-changing effects are applied.

Discreteness

Race conditions occur because of discretization of continuous systems

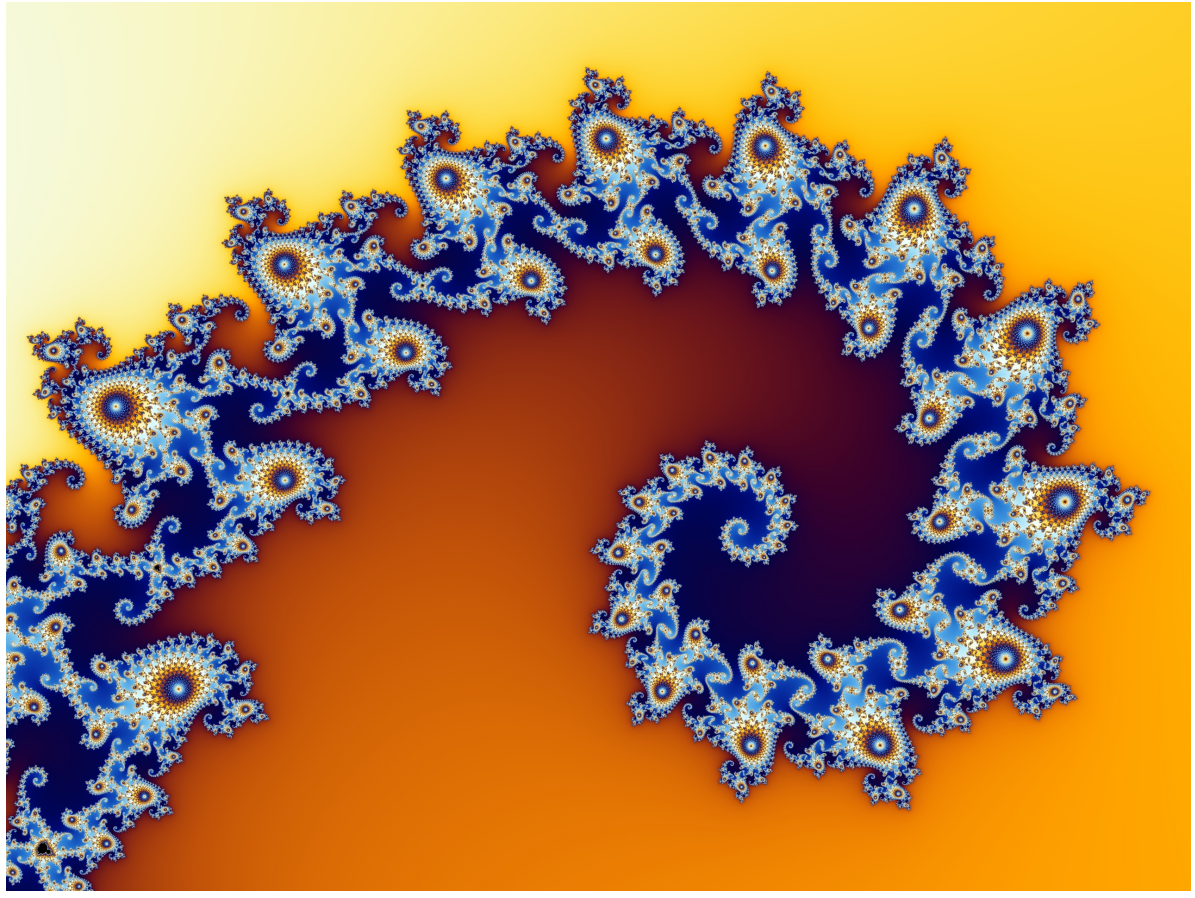
So why not deal in continuous systems?

Continuousness



In general, continuous systems lack representation

Continuousness



At best, we define processes then discretize them

Continuousness

Example

Select two random reals between 0 and 1

The probability of selecting the same real both times is 0

$$P = \lim_{x \to \infty} \frac{1}{x} = 0$$

Continuousness

How does one select a random real?

We could define an algorithm, but it's nonterminating

So this example lacks implementation

Referential Transparency

A pure expression can be replaced with its result without affecting the program

This lacks implementation as well

Referential Transparency

Referential transparency is the single-write strategy

In other words, dependency elimination so no race conditions

Referential Transparency

Referential transparency is the single-write strategy

In other words, dependency elimination so no race conditions

What should we do with mutable state?

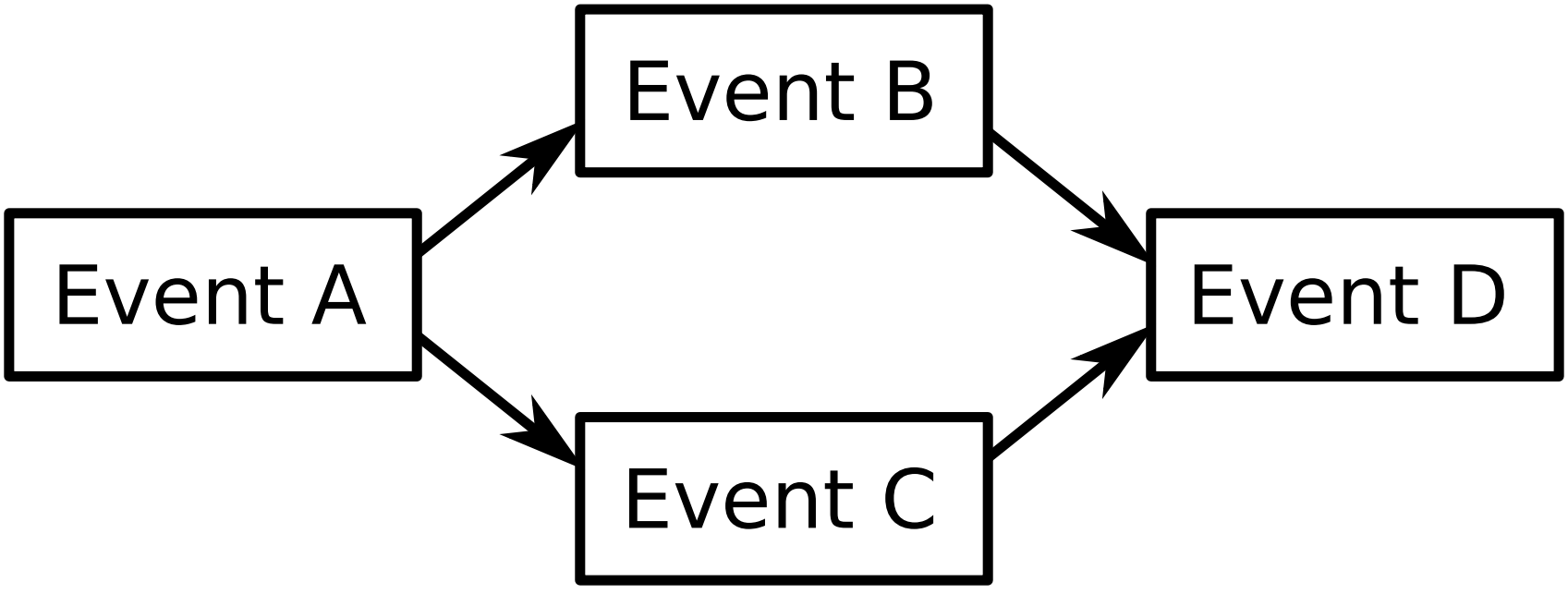
Dependency Graphs

RIO

In discrete time, order isn't defined below temporal resolution

This is how we get race conditions

Dependency Graphs



Dependency Graphs

- Language syntax should be isomorphic to dependency edges

  statements of the form
  (event)
  (stateful reads and pure transforms)
  (write target)

Dependency Graphs

- Language syntax should be isomorphic to dependency edges

  statements of the form
  (event)
  (stateful reads and pure transforms)
  (write target)

- Compiler should forbid diamond patterns
- Requires language-level implementation
- Style should adapt accordingly

Dependency Graphs

Is this too strict?

Is this scalable?

What concessions can we make? What other nodes/subgraphs are safe?

We need a language first...

Conclusion

Race conditions are the biggest problem

Referential transparency solves huge classes of them

Language-level dependency analysis may work

Conclusion

http://rio-lang.org

sjw@imap.cc

Follow me to B115 for more!