Lecture 13
Z Examples

David Garlan
Carnegie Mellon University

Models of Software Systems
Fall 2016

The Story Thus Far

- We have seen a brief introduction to Z
- Main features
  > use of schemas to represent state

  > use of a schema to represent initial states

  > use of schemas to describe operations
    - » the delta and xi conventions
    - » ? and ! conventions

  > use of schema calculus to build up bigger operations out of smaller ones
    - » separate normal and error behaviors

  > use of framing schemas and promotion
    - » separate behavior of an individual from a collection
    - » simplifies model maintenance

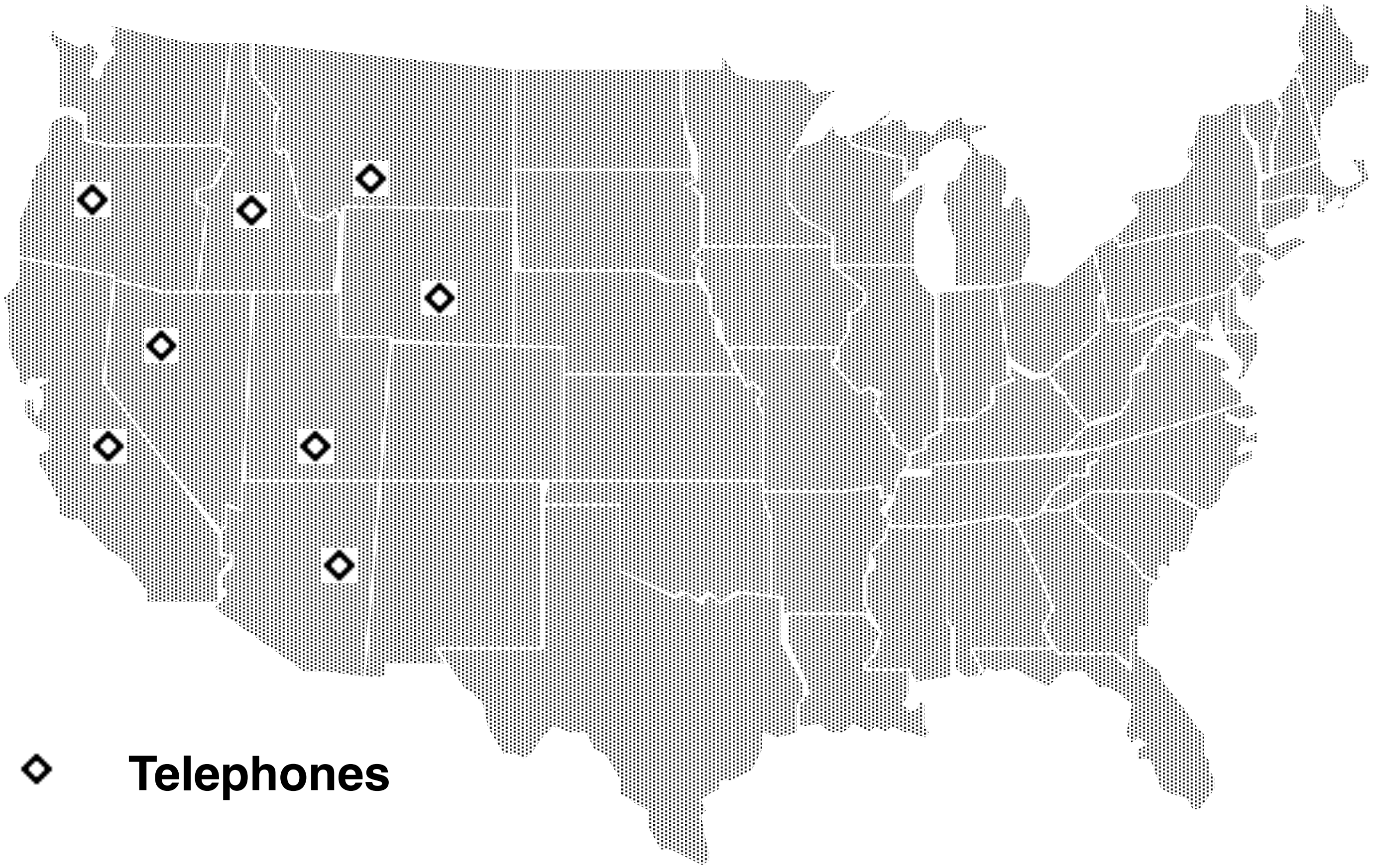
This Lecture

- Bridging the gap between theory and practice
- Part 1: The Telephone Net Problem
  > Another example of the schema calculus
- Part 2: Z Specification of the WSDL Standard
  > Created by the W3C to explain the Web Services Description Language
- Part 3: Oscilloscopes
  > *A Formal Specification of an Oscilloscope*, Norman Delisle and David Garlan. IEEE Software Sept. 1990

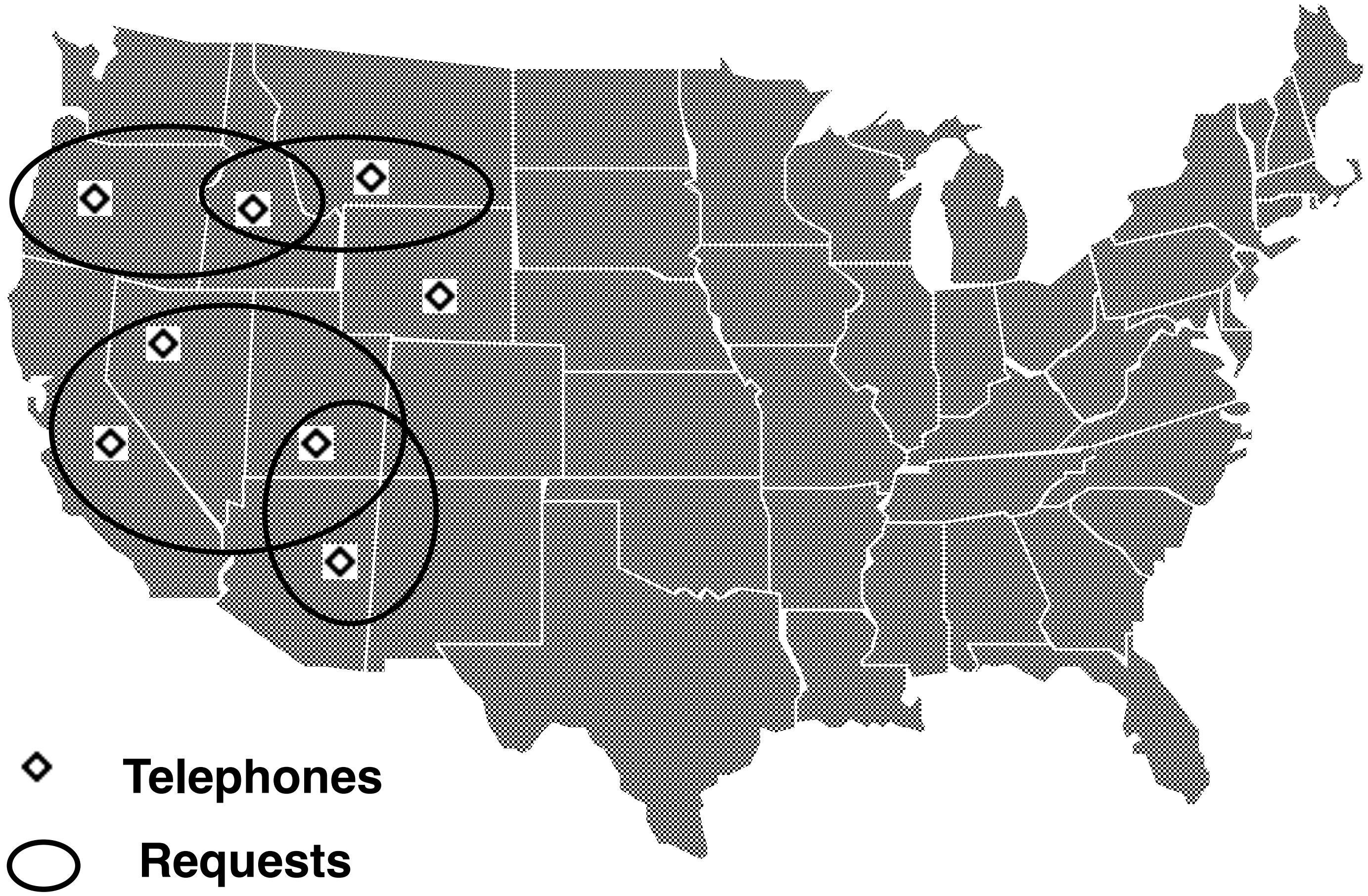
Telephone Net

- **Telephone Net**
  - > Adapted from "Telephone Network", by Carroll Morgan
  - > In Specification Case Studies, Ian Hayes (ed.)
- **Concepts**
  - > Use of strong state invariants to make concise models
  - > Use of schema calculus to simplify descriptions and associated reasoning
  - > Disj (Generic Definitions)
  - > Framing
  - > Reasoning with state invariants

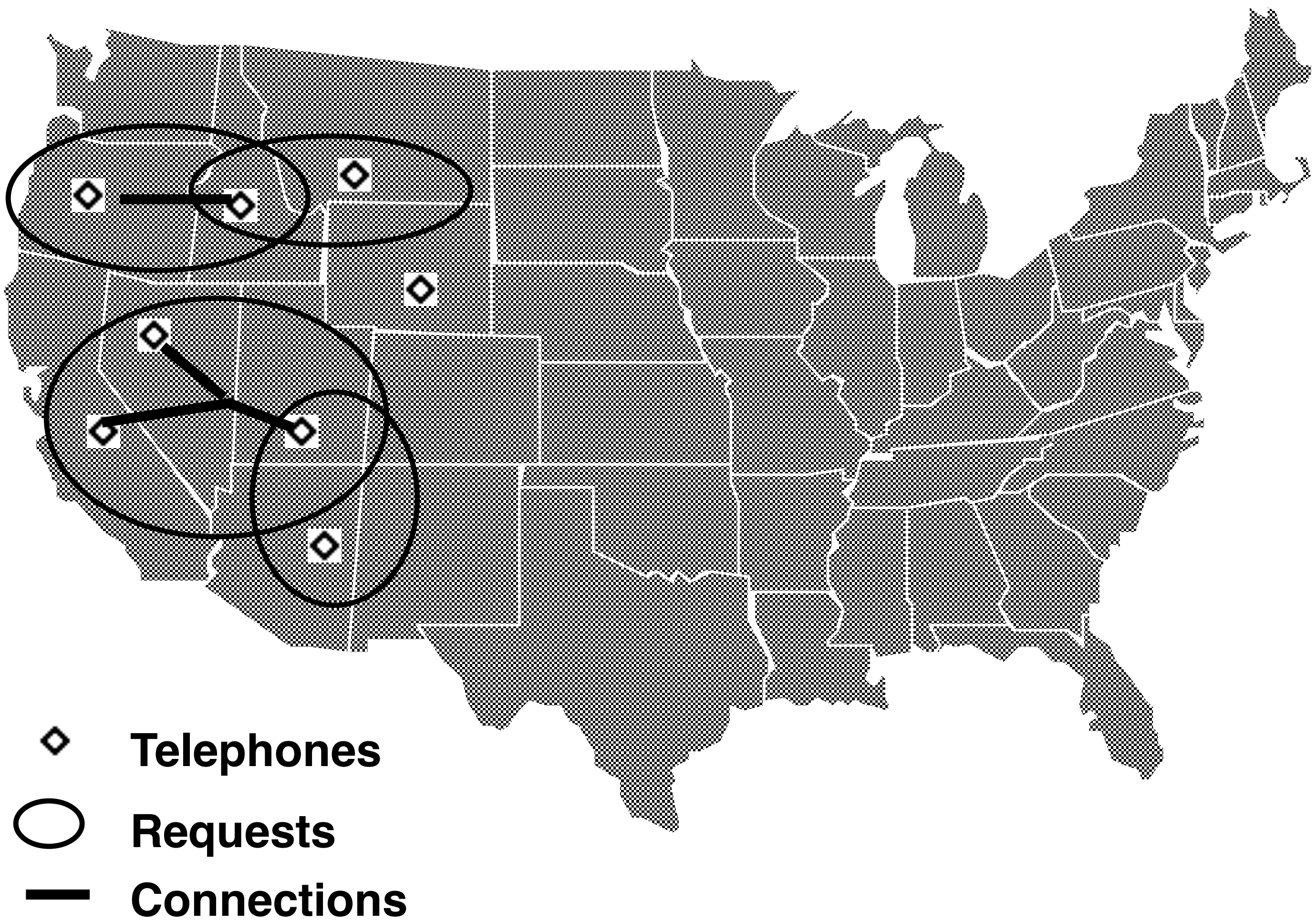
Telephones



Requests for Connection



Connections



Operations

- ***Call*: Request a connection between two phones**
- ***Hangup*: Terminate a connection**
- ***Busy*: Report whether a phone is busy**

State Space

[PHONE]

CONNECTION == $\mathbb{P}$ PHONE

TelephoneNet

reqs, cons: $\mathbb{P}$ CONNECTION

cons $\subseteq$ reqs

disj cons

Example

PHONE = {1, 2, 3 }

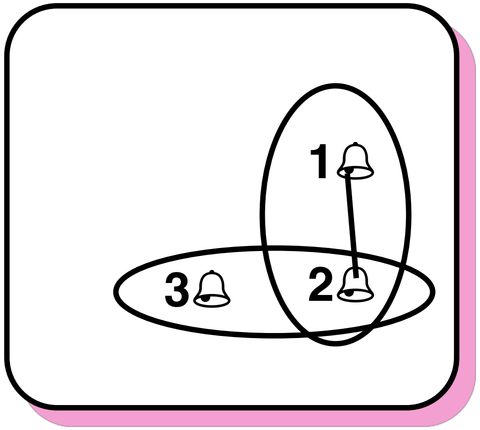
CONNECTIONS =

{ {}, {1}, {2}, {3}, {1,2} {1,3}, {2,3}, {1,2,3} }

TelephoneNet

reqs: { {1,2},{2,3} }

cons: { {1,2} }



Pairwise disjoint sets

Definition: (informal) A set of sets is disjoint iff no pair of those sets has elements in common.

Definition: (formal)

disj cons $\Leftrightarrow$

$\forall$ c1, c2 : cons • c1 $\neq$ c2 $\Rightarrow$ c1 $\cap$ c2 = $\varnothing$

Examples: Which sets are *disj*?

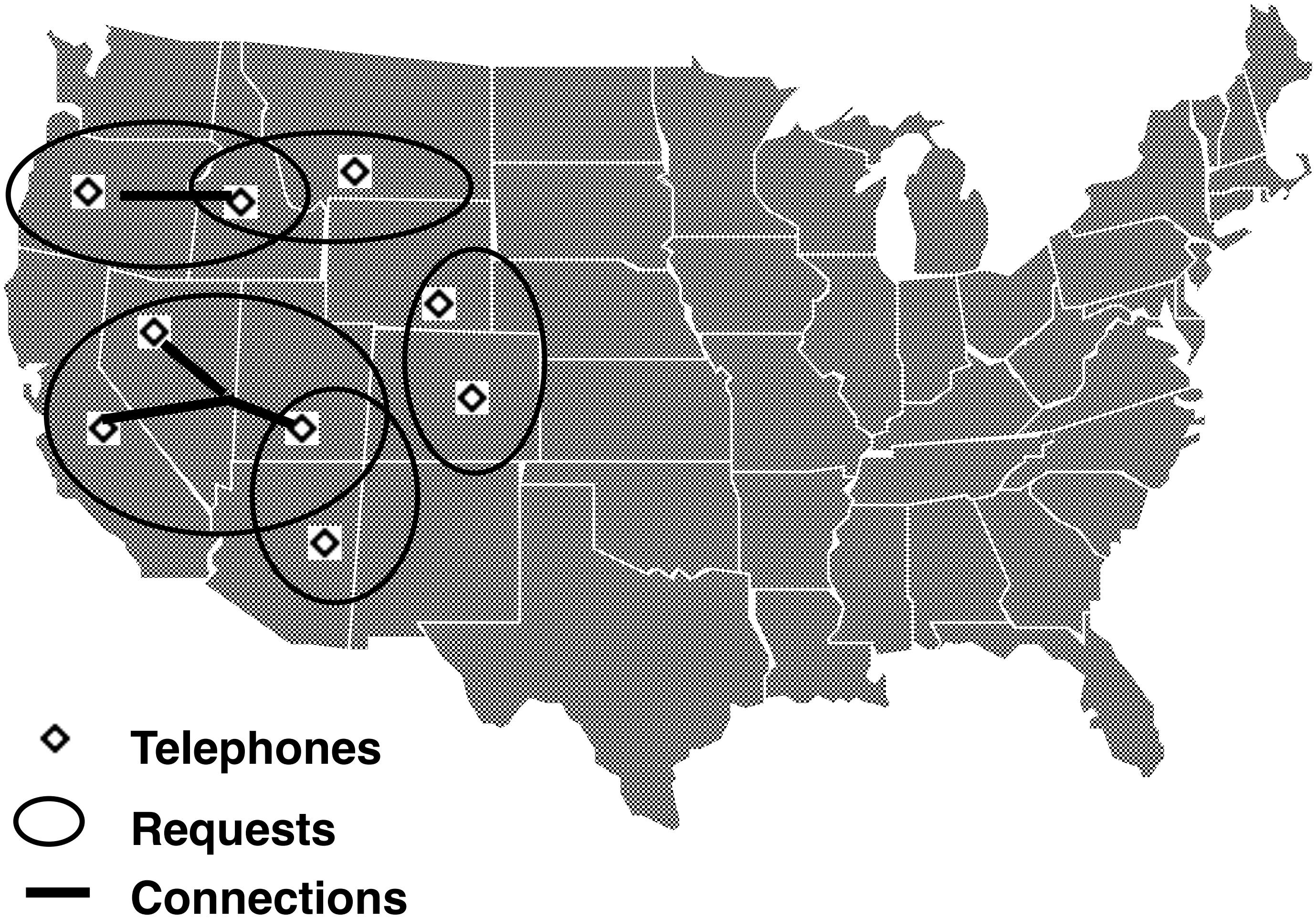
{ {a}, {b,c}, {d} }

{ {a,b}, {c,d}, {a,c} }

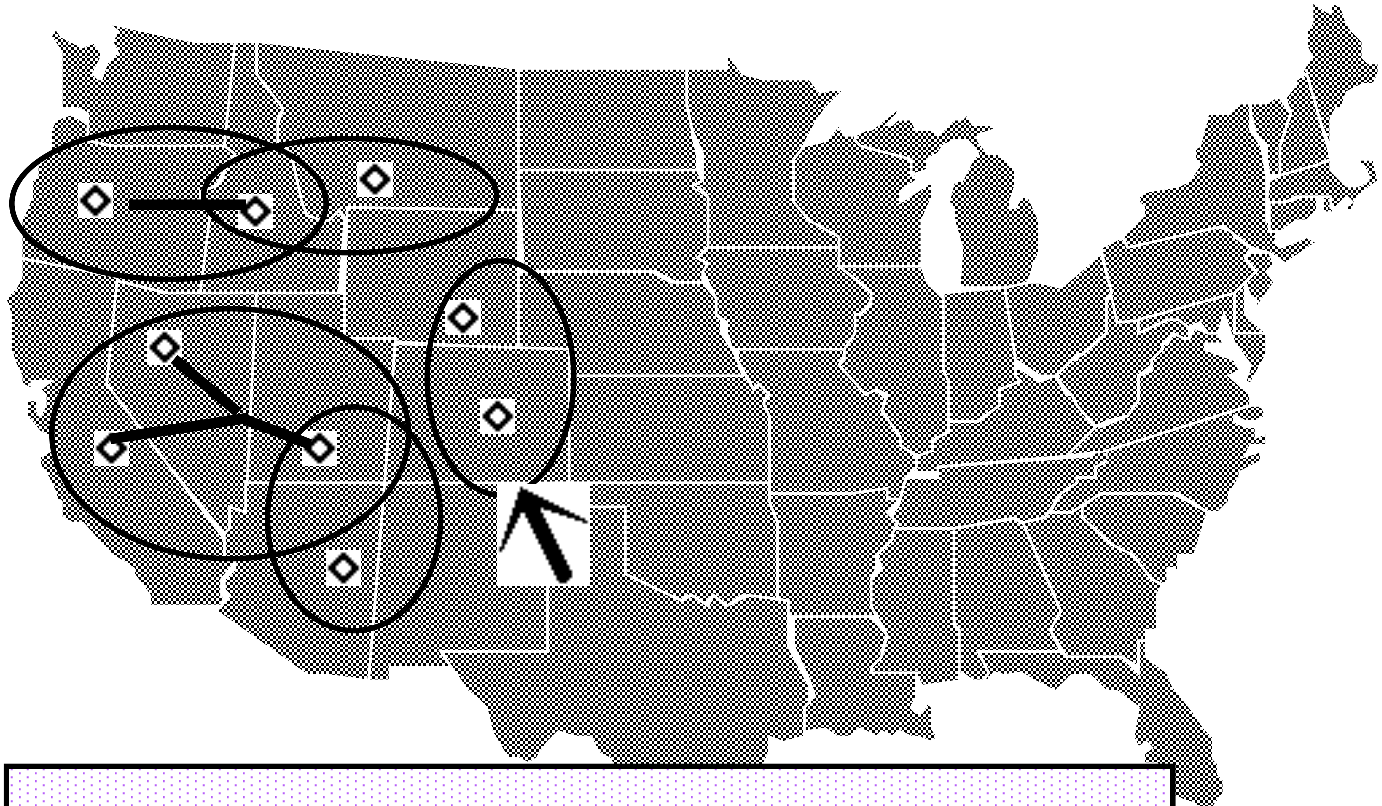
{ $\varnothing$, {a} }

{ {a,b}, {b, a}}

What's Wrong With This Net?



An Inefficient Net



IneffiecientNet :

$\exists$ r: reqs \ cons • disj (cons $\cup$ {r})

Efficient Networks

EfficientTN

TelephoneNet

$\sim (\exists\ r: reqs \setminus cons \bullet disj\ (cons \cup \{r\}))$

EfficientTN

reqs, cons: $\mathbb{P}$ CONNECTION

---

cons $\subseteq$ reqs

disj cons

~ ($\exists$ r: reqs \ cons • disj (cons $\cup$ {r}))

Initial Telephone Net

InitTN

EfficientTN

reqs = cons = ∅

Framing: Event

Event

Δ **EfficientTN**

$\forall$ **c: CONNECTION •**

$c \in (cons \cap reqs') \Rightarrow c \in cons'$

This Really Means ...

Event

reqs, cons: $\mathbb{P}$ CONNECTION

reqs', cons': $\mathbb{P}$ CONNECTION

---

cons $\subseteq$ reqs

disj cons

cons' $\subseteq$ reqs'

disj cons'

~($\exists$ r: reqs \ cons • disj (cons $\cup$ {r}))

~($\exists$ r: reqs' \ cons' • disj (cons' $\cup$ {r}))

$\forall$ c: CONNECTION • c $\in$ (cons $\cap$ reqs') $\Rightarrow$ c $\in$ cons'

But we would not be expected to write it out this way.

Call

Call

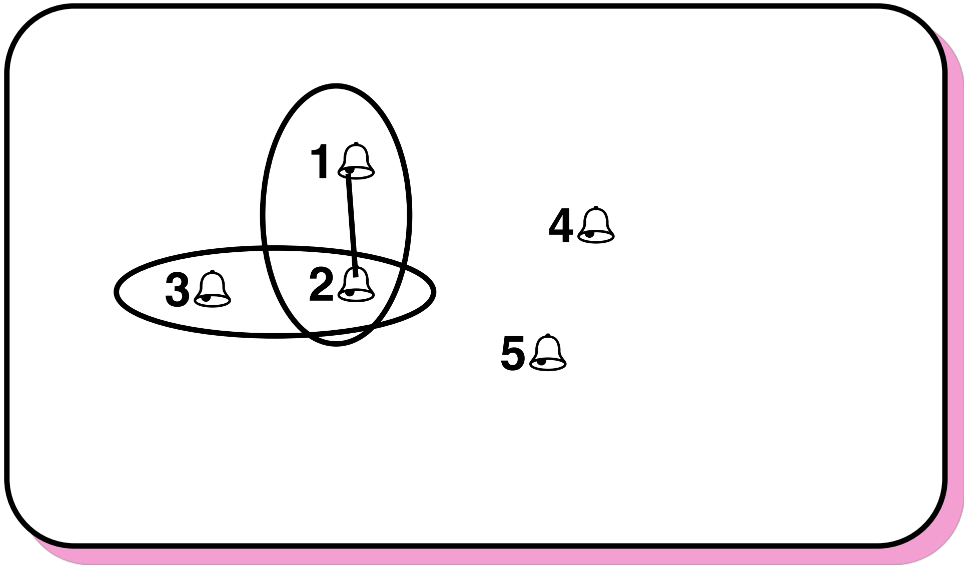
Event

ph?, dialled? : PHONE

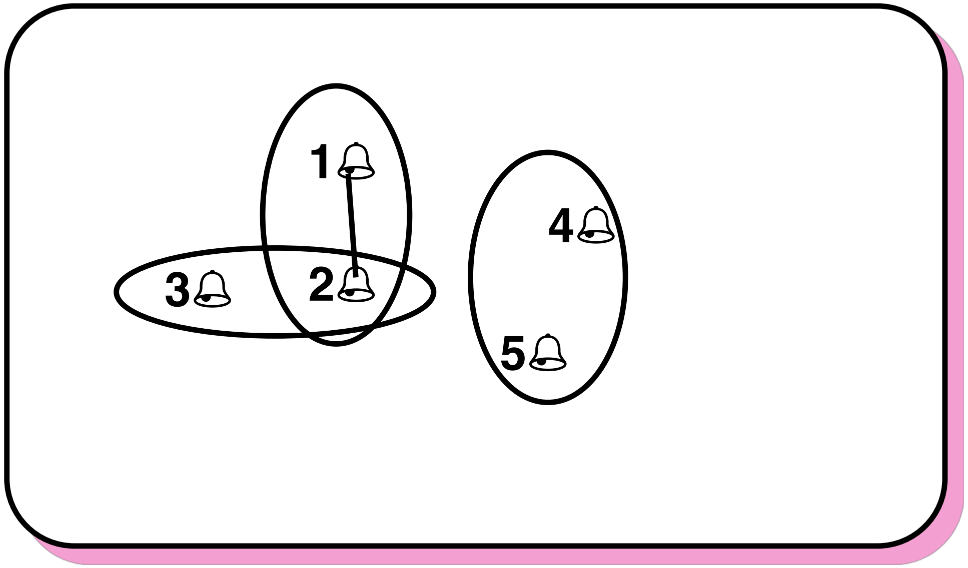
---

reqs' = reqs $\cup$ { {ph?, dialled?} }

Example



Example



Must the new request become a connection?

Hangup

Hangup

Event

ph? : PHONE

$$reqs' = reqs \setminus \{ c: cons \mid ph? \in c \}$$

Is this realistic?

Busy

YesOrNo ::= Yes | No

$$
\begin{array}{|l}
\textbf{Busy} \\
\hline
\text{Event} \\
ph?: PHONE \\
busy!: YesOrNo \\
\hline
reqs' = reqs \\
busy! = Yes \Leftrightarrow ph? \in \bigcup cons \\
\hline
\end{array}
$$

What happens to cons?

Reasoning about the Specification

- **Theorem**: (Informal) If an operation doesn't change any of the requests then it doesn't change any of the connections.
- **Proof**: (Informal)
  1. The constraint on Event tells us that any original connection won't be broken if it is still requested.
  2. If an operation doesn't change the requests, all original connections will still be requested.
  3. Hence, no connections are broken.
  4. Also, no new connections are added because if a connection could be added afterwards, it could have been added before.
  5. So connections aren't changed.

Part 2: Formally Specifying the WSDL Standard

- **Sources:**
  - > "Adventures in Formal Methods at W3C: Using Z Notation to Specify WSDL 2.0," ***Arthur Ryman, IBM***
    http://www.w3.org/2006/Talks/0301-z-notation.pdf
  - > The WSDL Specification
    http://www.w3.org/TR/wsdl20/wsdl20-z.html

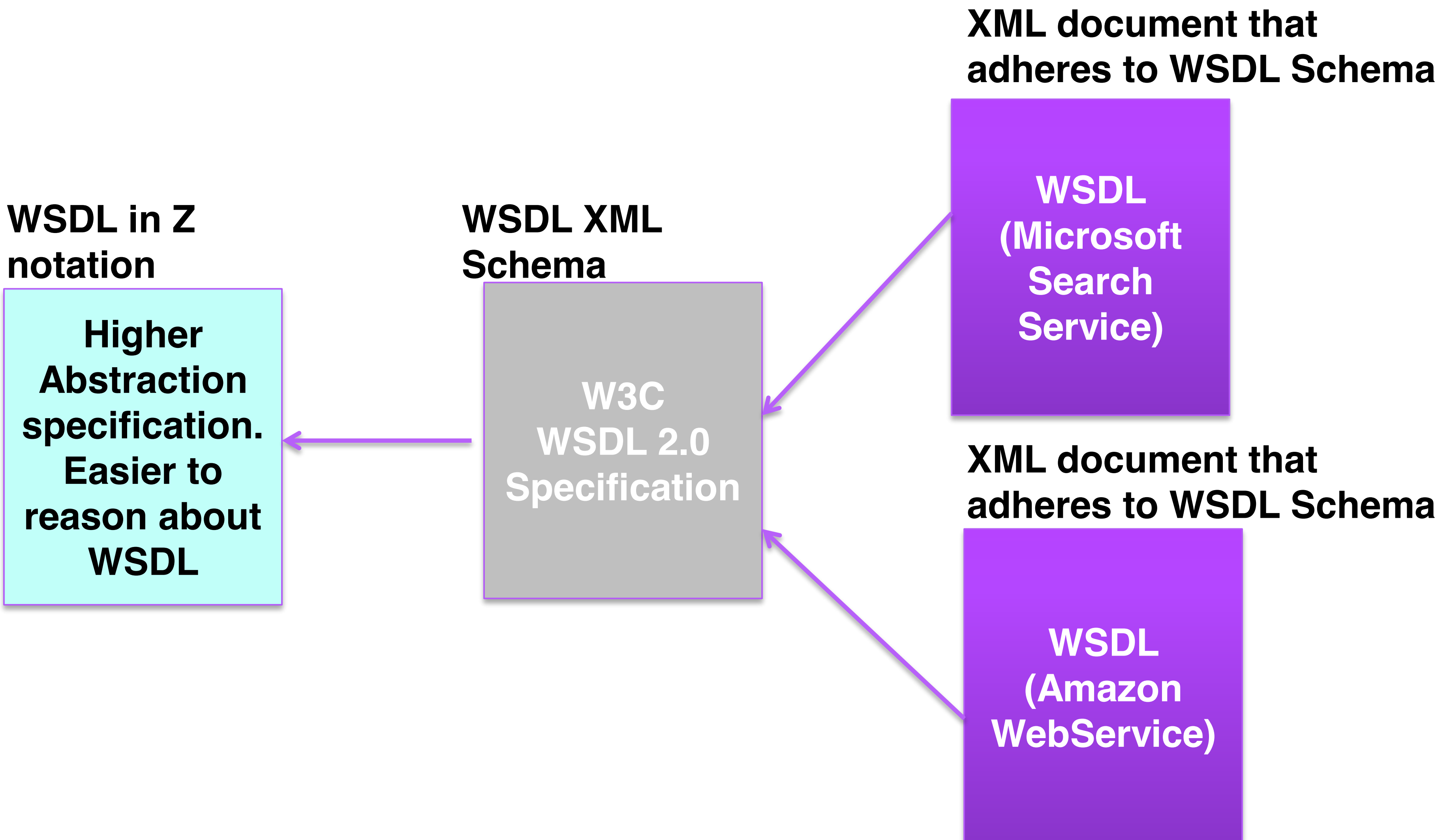
Web Services Description Language (WSDL)

- WSDL is a standard used to describe Web services
  - > Developed by the World Wide Web Consortium (W3C)
  - > Describes the interface to a service that can be called using a web service protocol (such as SOAP)
  - > Is also used to locate web services
- A WSDL definition is written in XML
  - > WSDL valid form is defined by an XML Schema

Example: Microsoft MSN Search

```
<?xml version="1.0" encoding="utf-8" ?>
- <wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:wsdl="http://schemas.xm
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing" xmlns:tns="http://s
    targetNamespace="http://schemas.microsoft.com/MSNSearch/2005/09/fex">
  + <wsdl:types>
  - <wsdl:message name="SearchMessage">
      <wsdl:part name="parameters" element="tns:Search" />
    </wsdl:message>
  - <wsdl:message name="SearchResponseMessage">
      <wsdl:part name="parameters" element="tns:SearchResponse" />
    </wsdl:message>
  - <wsdl:portType name="MSNSearchPortType">
    - <wsdl:operation name="Search">
        <wsdl:input message="tns:SearchMessage" wsa:Action="http://schemas.microsoft.co
        <wsdl:output message="tns:SearchResponseMessage"
          wsa:Action="http://schemas.microsoft.com/MSNSearch/2005/09/fex/MSNSearch
      </wsdl:operation>
    </wsdl:portType>
  - <wsdl:binding name="MSNSearchPortBinding" type="tns:MSNSearchPortType">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
    - <wsdl:operation name="Search">
        <soap:operation soapAction="http://schemas.microsoft.com/MSNSearch/2005/09/fe
      + <wsdl:input>
      + <wsdl:output>
      </wsdl:operation>
    </wsdl:binding>
  - <wsdl:service name="MSNSearchService">
    - <wsdl:port name="MSNSearchPort" binding="tns:MSNSearchPortBinding">
        <soap:address location="http://soap.search.live.com:80/webservices.asmx" />
      </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Problem: XML Schema is Complex



Conceptual Model

- A WSDL 2.0 document, and its related documents, defines a set of components that together form an instance of a Component Model.
- This specification defines the structure and constraints on the components in a valid component model instance.

Motivation for a Formal Specification

- Ryman and a team in the W3C, specified the WSDL 2.0 Schema using an abstract informal model.
  > "Component Model" inspired by the XML Infoset and schema specifications
- This specification was long and difficult to keep consistent
  > Needed to enforce component relationships and properties across the specification

Nature of the Specification

- Interesting parts of the specification.
  - Check for uniqueness of the componentIDs
  - Used schema calculus to build the complete WSDL document abstraction.
- Notes:
  - There are no operations (Δ).
  - They are modeling a specification, not a machine with behavior. Therefore no transitions.
  - The specification was type checked with Fuzz

Example: Uniqueness of Components

A component model is a set of uniquely identified components that satisfy a set of validity constraints

Components **is the set of components in the component model**
componentIds **is the set of identifiers of components in the component model**

ComponentModel1

$$
\begin{array}{l}
components : \mathbb{P}\, Component \\
componentIds : \mathbb{P}\, ID \\
\hline
\forall x, y : components \bullet \\
\qquad Id(x) = Id(y) \Rightarrow x = y \\
componentIds = \{ x : components \bullet Id(x) \}
\end{array}
$$

Example: Validate Duplicate

▼ InterfaceKey [show all] [hide all]

Let *InterfaceKey* express the QName uniqueness constraint on the Interface component:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\textit{InterfaceKey}\hspace{10em}}} \\
\textit{ComponentModel2} \\
\hline
\forall x, y : \textit{interfaceComps} \mid \\
\quad x.\textit{name} = y.\textit{name} \bullet x = y \\
\hline
\end{array}
$$

See *ComponentModel2*.

- No two Interface components have the same {name} property.

Example: Components Names

Let *ElementDeclarationCM* express this constraint:

$$
\begin{array}{|l}
\textit{ElementDeclarationCM} \\
\hline
\textit{ComponentModel2} \\
\hline
\forall x, y : \textit{elementDeclComps} \mid \\
\quad x.name = y.name \land \\
\quad x.system = y.system \bullet \\
\quad\quad x = y \\
\hline
\end{array}
$$

See *ComponentModel2*.

- No two Element Declaration components have the same {name} and {system} properties.

Using the Schema calculus

Let *ComponentModel2* be the basic component model, augmented with the definitions of the subsets of each component type and their corresponding identifiers

$$
\begin{aligned}
&\mathit{ComponentModel2} \mathrel{\widehat=} \\
&\qquad \mathit{InterfaceComponentIds} \land \\
&\qquad \mathit{BindingComponentIds} \land \\
&\qquad \mathit{ServiceComponentIds} \land \\
&\qquad \mathit{OtherComponentIds}
\end{aligned}
$$

Part 3: Oscilloscopes

The Problem

At Tektronix

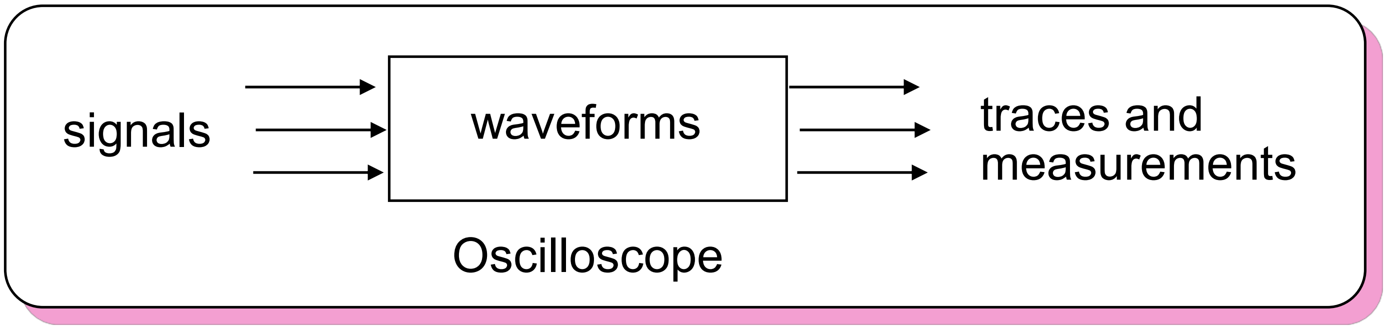
- Many divisions building similar products
- No reuse among product efforts
- Long time-to-market (5 years)
- Each product typically built from scratch
- User interface increasingly important
- Software effort increasingly larger proportion of development costs

Goals

- Build next generation instrumentation systems
- Allow reuse between product divisions
- Support better interactive response to user changes
- Multiple hardware platforms for same user interface
- Multiple user interfaces for same platform (vertical markets)

What is an Oscilloscope?

Oscilloscopes are simple in theory



But complex in practice

- Complex software
- Multi-processing environment
- Special hardware
- Sophisticated user interface
- Many specialized modes

Reminder: Z Notation thus far

- **Given sets:**

  [DATE, TITLE]

- **Axiomatic schemas**

  $$\begin{array}{|l} f: \mathbb{Z} \rightarrow \mathbb{Z} \\ \hline \forall x: \mathbb{Z} \bullet f(x) = x^2 \end{array}$$

  $$\begin{array}{|l} MAX: \mathbb{N} \end{array}$$

- **Enumerations**

  REPORT ::= ok | already-known | not-known

- **Schemas**

  $$\begin{array}{|l} \text{\_\_ SchemaName \_\_\_\_\_\_\_\_} \\ \text{declarations} \\ \hline \text{state invariant} \\ \hline \end{array}$$

Defining Functions in Z

- We define a function by giving the rule for calculating its values
- For example, to characterize the "square function" informally we might write

  $f(x) = x^2$

- We use the following Z notation to do this

  $f: A \rightarrow B$

  predicate over f

- So in this example we would say

  $f: \mathbb{N} \rightarrow \mathbb{N}$

  $\forall x: \mathbb{N} \bullet f(x) = x^2$ or $\forall x: \mathbb{N} \bullet f\ x = x^2$

Alternative Function Definition: Lambda Notation

- We can also use “lambda notation”
- The general form is

  $f = \lambda x: T \bullet \text{expression}$

- In this example

  $f = \lambda x: \mathbb{N} \bullet x^2$

- May also use multiple variables

  $plus = \lambda x,y: \mathbb{N} \bullet x + y$

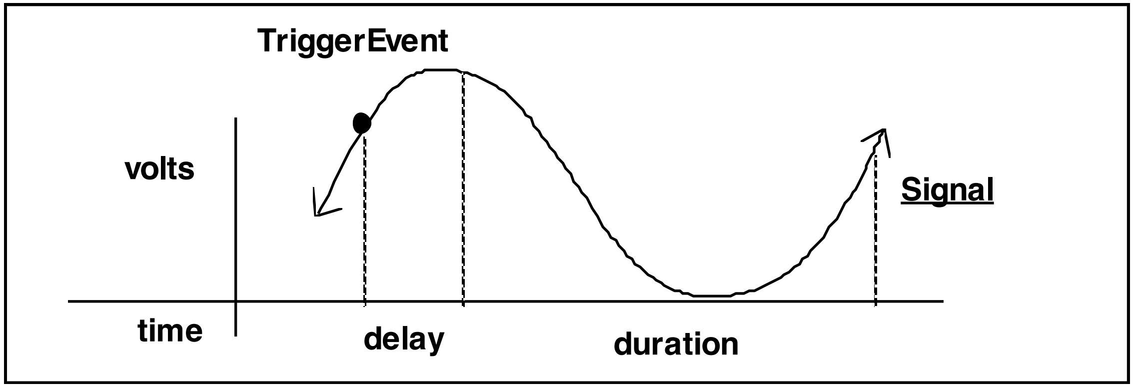
Oscilloscope: Functional View

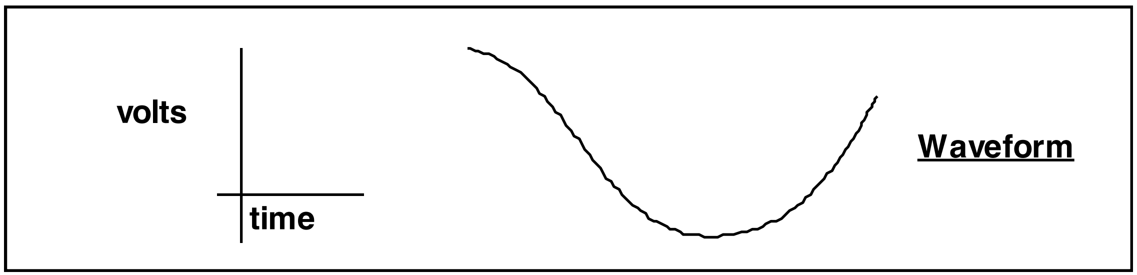
Traces = Oscilloscope (Signals)

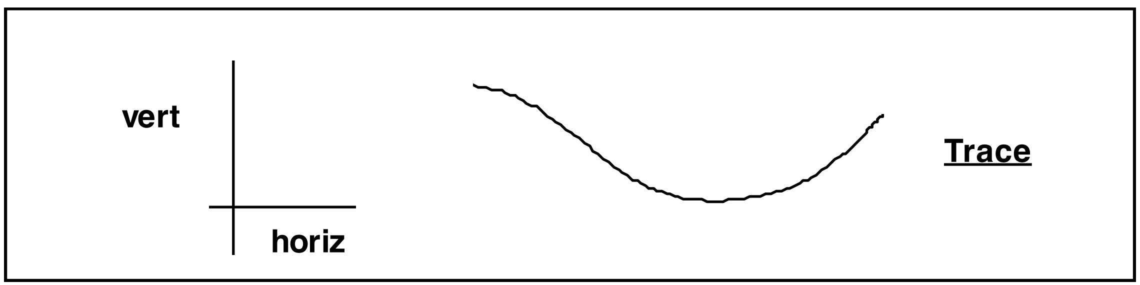
<u>But</u>:

1. **How to handle user input?**
2. **How can you decompose it into manageable pieces?**

Signals, Waveforms, Traces







Basic Types

AbsTime $== \mathbb{N}$

RelTime $== \mathbb{N}$

Volts $== \mathbb{Z}$

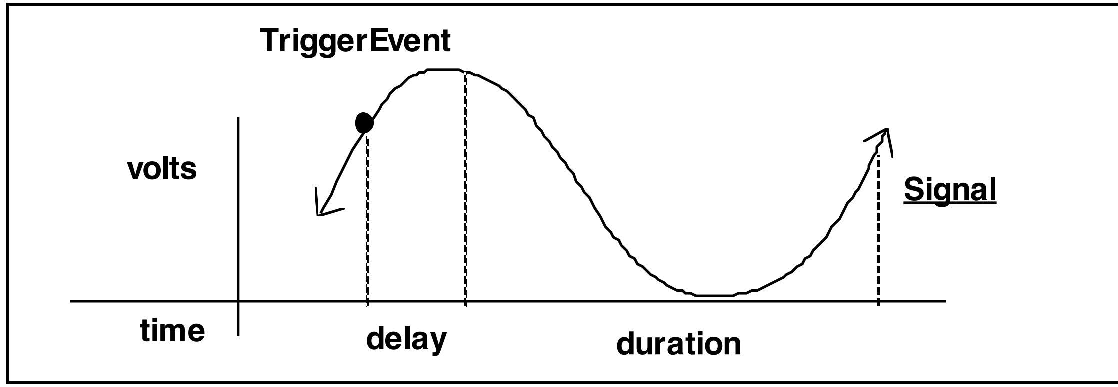
Horiz $== \mathbb{Z}$

Vert $== \mathbb{Z}$

Signals, Waveforms, Traces

Signal ==

AbsTime $\rightarrow$ Volts

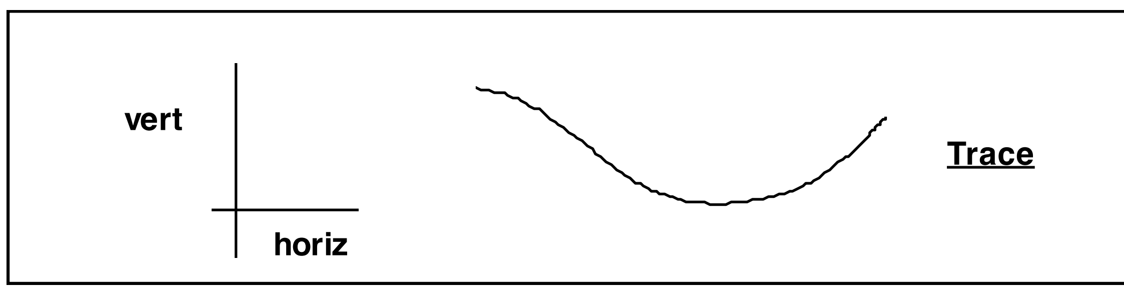


Waveform ==

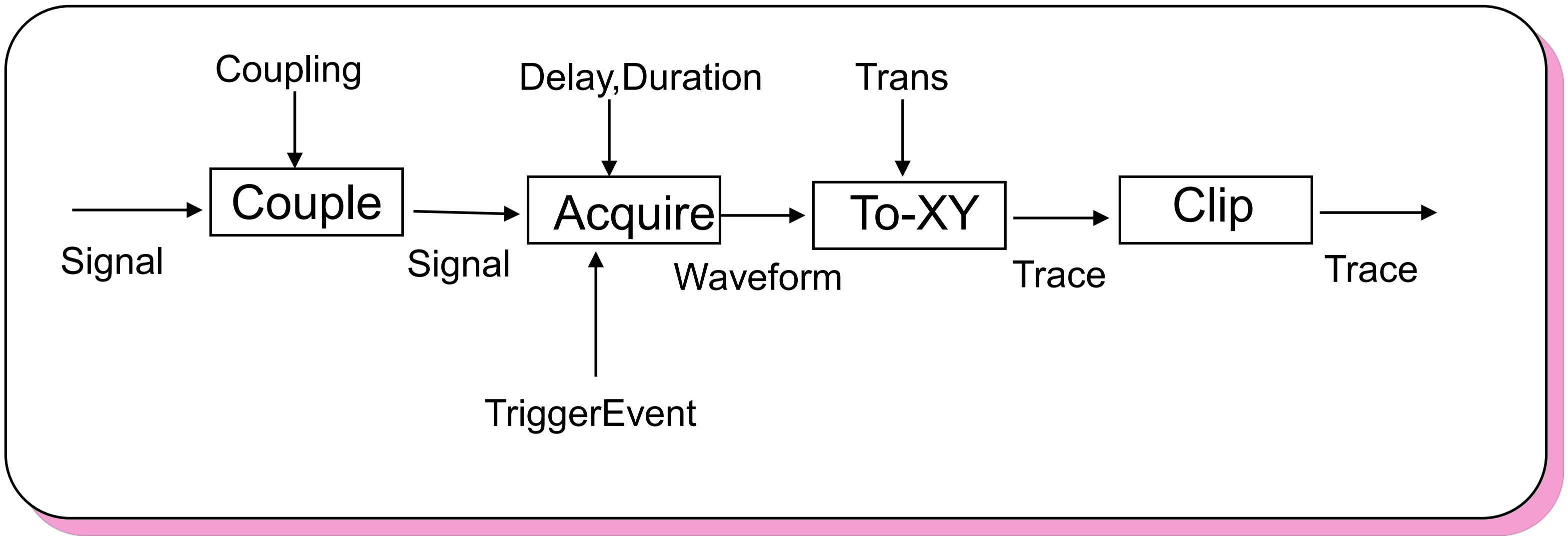
AbsTime $\nrightarrow$ Volts

Trace ==

Horiz $\rightarrow$ Vert



Channel Subsystem



Coupling

Coupling ::= DC | AC | GND

$$
\begin{array}{|l}
\text{Couple: Coupling} \rightarrow \text{Signal} \rightarrow \text{Signal} \\
\text{dc: Signal x AbsTime} \rightarrow \text{Volts} \\
\hline
\text{Couple DC s = s} \\
\text{Couple AC s =} \\
\qquad \lambda\ t\text{: AbsTime} \bullet s(t) - dc\ (s, t) \\
\text{Couple GND s =} \\
\qquad \lambda\ t\text{: AbsTime} \bullet 0
\end{array}
$$

Acquisition

TriggerEvent == AbsTime

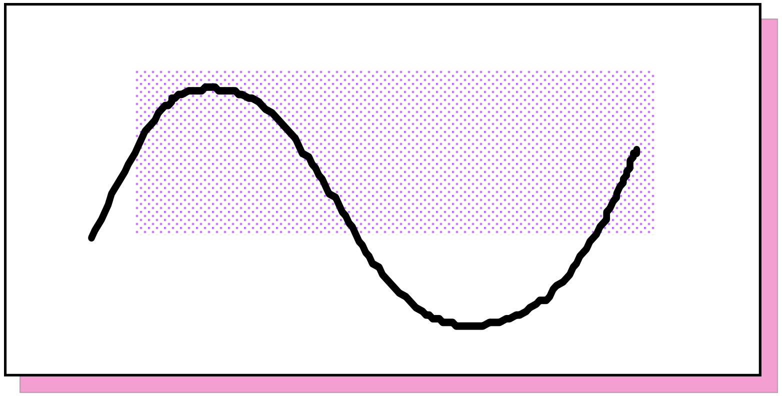
Acquire: RelTime x RelTime $\rightarrow$ TriggerEvent $\rightarrow$ Signal $\rightarrow$ Waveform

---

Acquire (delay, dur) trig s =

$\{ t: AbsTime \mid trig + delay \leq t \leq trig + delay + dur \} \lhd s$

Clip



maxH: Horiz

maxV: Vert

Clip : Trace → Trace

Clip tr = (0 .. maxH) ◁ tr ▷ (0 .. maxV)

Channel Parameters

Channel Parameters

c: Coupling

delay, duration: RelTime

scaleH: RelTime

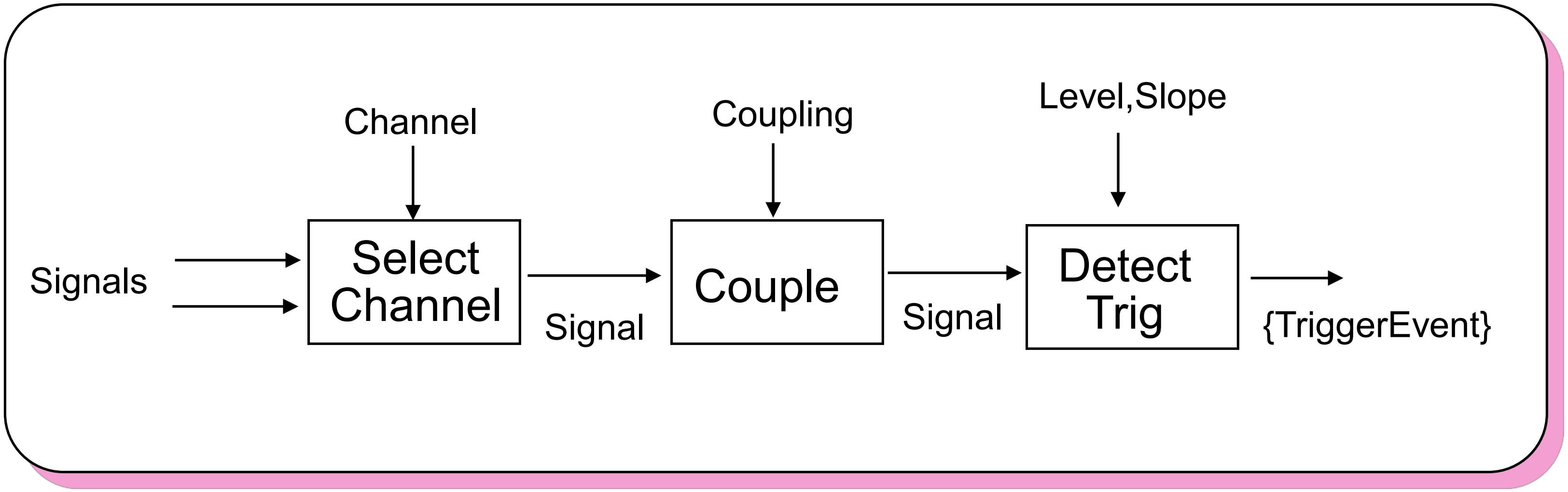
scaleV: Volts

posnV: Horiz

Channel Configuration

$$
\begin{array}{l}
\text{ChannelConfiguration: ChannelParameters} \rightarrow \\
\quad \text{TriggerEvent} \rightarrow \text{Signal} \rightarrow \text{Trace} \\
\hline
\text{ChannelConfiguration p =} \\
\quad \lambda \text{ trig: TriggerEvent} \bullet \\
\qquad \text{Clip} \circ \text{WaveformToTrace (...)} \circ \\
\qquad \text{Acquire (p.delay, p.dur) trig} \circ \text{Couple p.c}
\end{array}
$$

Trigger Subsystem



Channel Selection

Channel ::= CH1 | CH 2

SelectChannel: Channel →
Signal x Signal → Signal

SelectChannel Ch1 (s1, s2) = s1

SelectChannel Ch2 (s1, s2) = s2

Trigger Detection

Slope ::= POS | NEG

Level == Volts

$$\text{DetectTrig}: \text{Level} \times \text{Slope} \rightarrow \text{Signal} \rightarrow \mathbb{P}\ \text{TriggerEvent}$$

$$\forall\ t: \text{DetectTrig}\ (l, sl)\ s \bullet$$

$$(sl = POS) \Rightarrow s\ (t - 1) \leq l \leq s\ (t)$$

$$(sl = NEG) \Rightarrow s\ (t) \leq l \leq s\ (t - 1)$$

Trigger Parameters

TriggerParameters

l: Level

sl: Slope

ts: Channel

tc: Coupling

$tc \in \{AC, DC\}$

Trigger Configuration

TriggerConfiguration: TriggerParameters $\rightarrow$ Signal x Signal $\rightarrow \mathbb{P}$ TriggerEvent

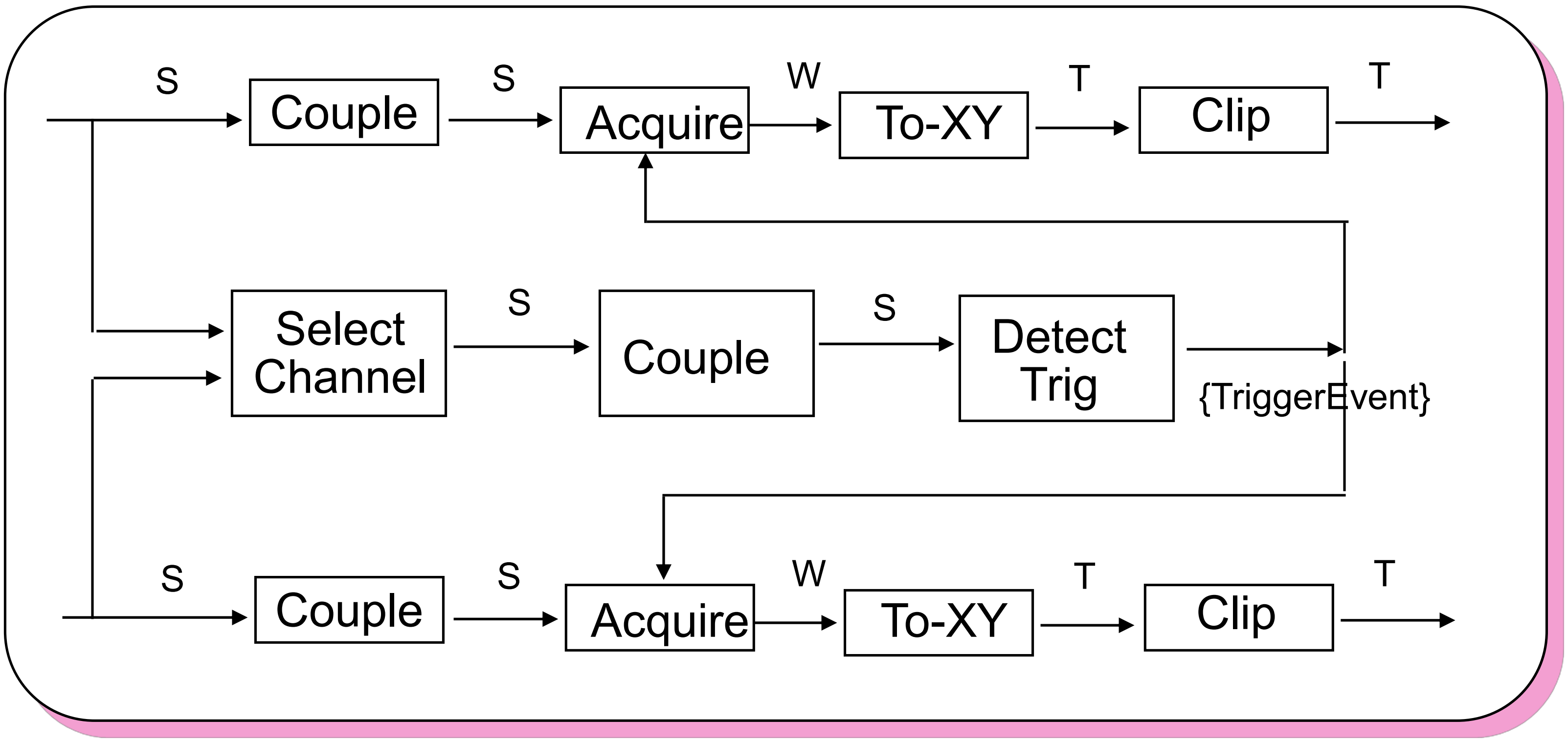
TriggerConfiguration p =

DetectTrig (p.l, p.sl) $\circ$ Couple p.tc $\circ$

SelectChannel p.ts

function composition

The Whole System



Oscilloscope

Oscilloscope

s1, s2: Signal

cp1, cp2: ChannelParameters

tp: TriggerParameters

ts1, ts2: seq Trace

∀ t: ran ts1 •

∃ trig: TriggerConfiguration tp (s1, s2) •

t = ChannelConfiguration cp1 trig s1

∀ t: ran ts2 • ...

Role of Formalism

- What was formalized
  - User level model (higher-order P/F)
  - Connection framework (colored P/F)
  - User interface architecture
- Benefits of formalism
  - Motivated and constrained architectures
  - Conceptual prototyping
  - Communication medium
- Non-benefits
  - Correctness
  - Completeness
  - Adoption of formal methods within Tektronix

Some Morals

- Success requires
  - Domain experts and system builders
  - Expertise in abstraction
  - Skill in manipulating formal models
  - Willingness to abandon old implementation patterns
  - Willingness to reject inappropriate models
- Tools
  - Not needed during conceptualization
  - Essential during development
- Management
  - Must keep hands off the process initially
  - Must help enforce standards later