EPFL

SWISS PLASMA CENTER

Centralised Event/State/Info Server (CESIS)

Basil P. DUVAL

For the TCV team, May 2019*

*Swiss Plasma Center, **EPFL***

Association EURATOM, Confédération Suisse

Lausanne, Switzerland

*First shown, March 2012, CRPP, Lausanne

Introduction

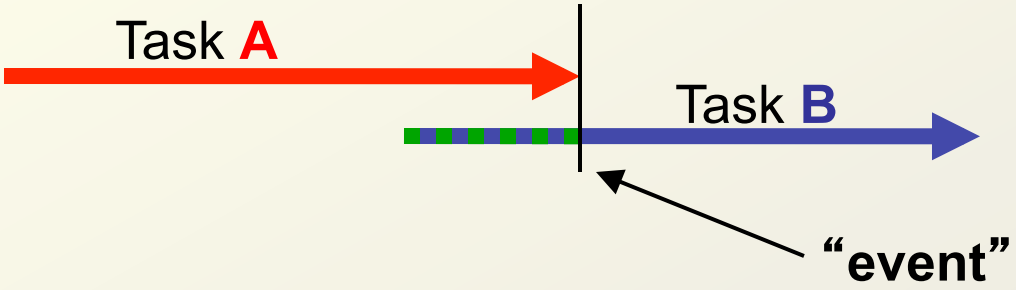
- Co-ordination of *many* processes (threads) across *many* machines
  ever **Popular / Needed**
- Many **IPC** (Inter-Process-Communication) layers -
  send messages (and/or data) between processes / machines, (sometimes continents)

- For an experiment, task **co-ordination** particularly important
  - **time** @ and/or between task launching often variable
  - depends on many **external** influences

- Task sequencing of a varied range of tasks often performed by a "**dependency**" table
  i.e. a particular task awaits certain "conditions"

- In a Clock-Real (i.e. experimental) world we often invoke the idea of "**events**"
  i.e. generated by one task, awaited by others

Example: an “event”



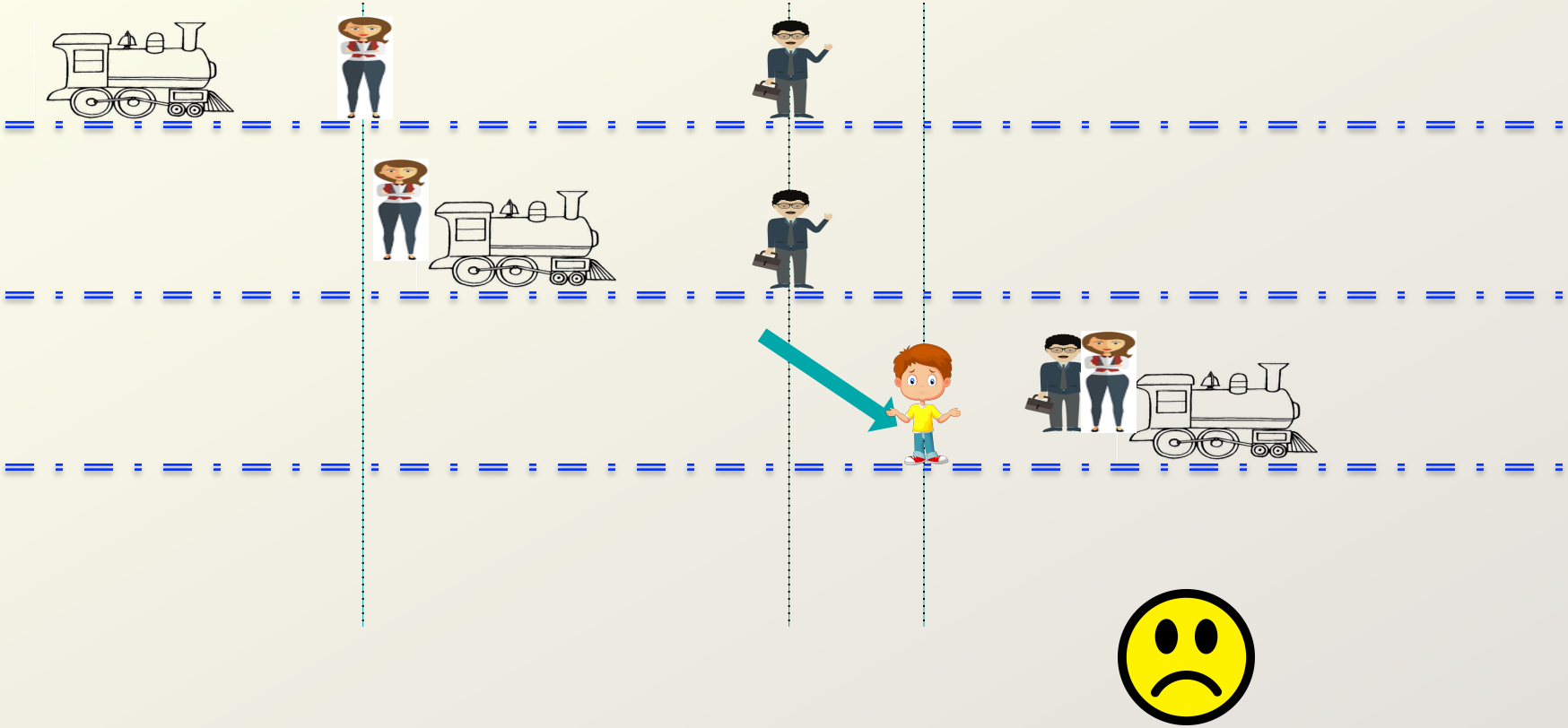
- Task **A** runs and creates “event”
- Task **B** runs before “event” and waits until “event” before executing. (inspired from MDSevents)

- “**event**” is at a **point** in time.

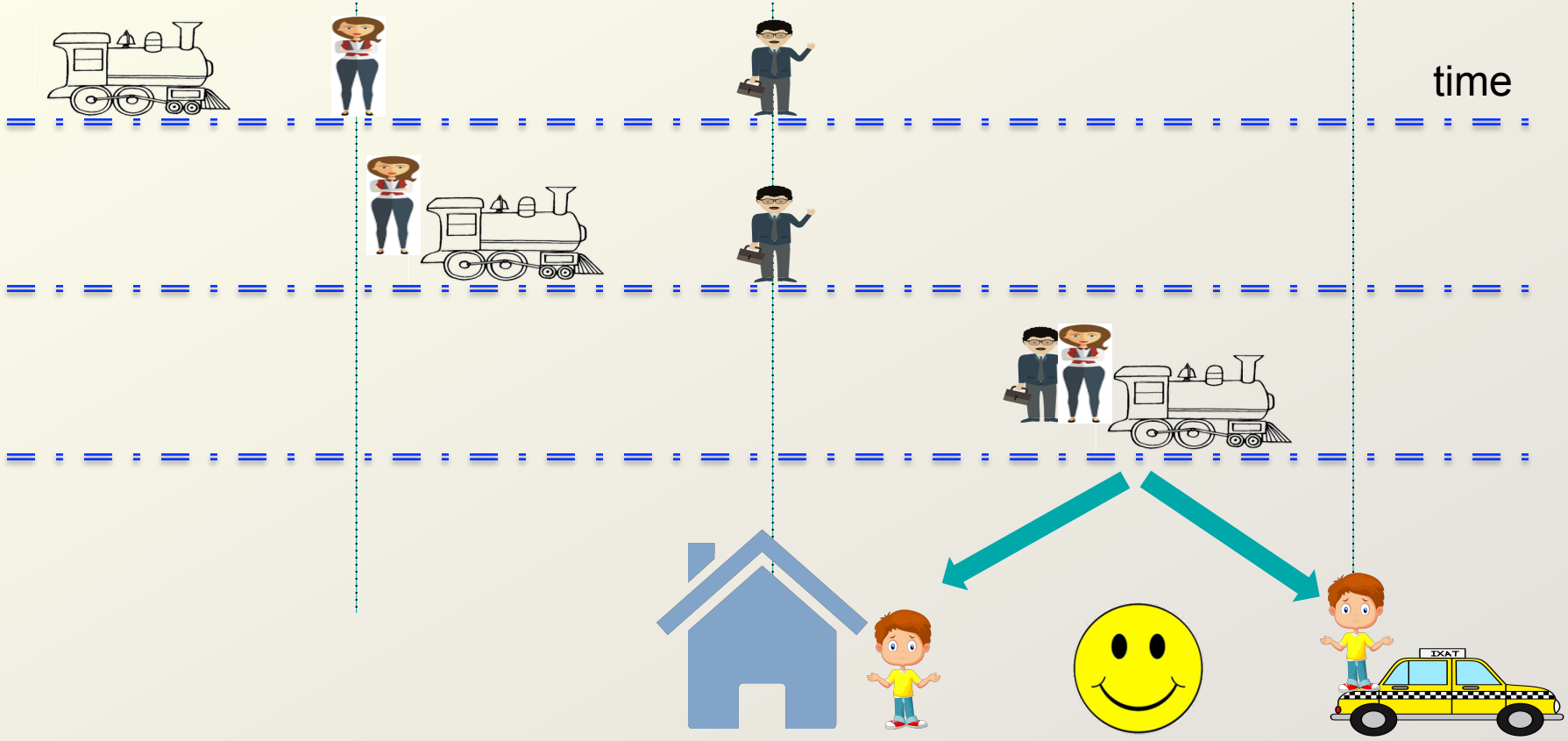
No transition !

a given time is either *before* or *after* an event.

The Train



The Train- corrected



Why something new ?

- Tasks should be able to ask:
  "has the event **already happened/passed**?"
- Tasks should **not "poll"** an event but be awoken by said event
- If you're waiting, you should, at your **own time**, be able to return to your program and decide whether to wait again

Also

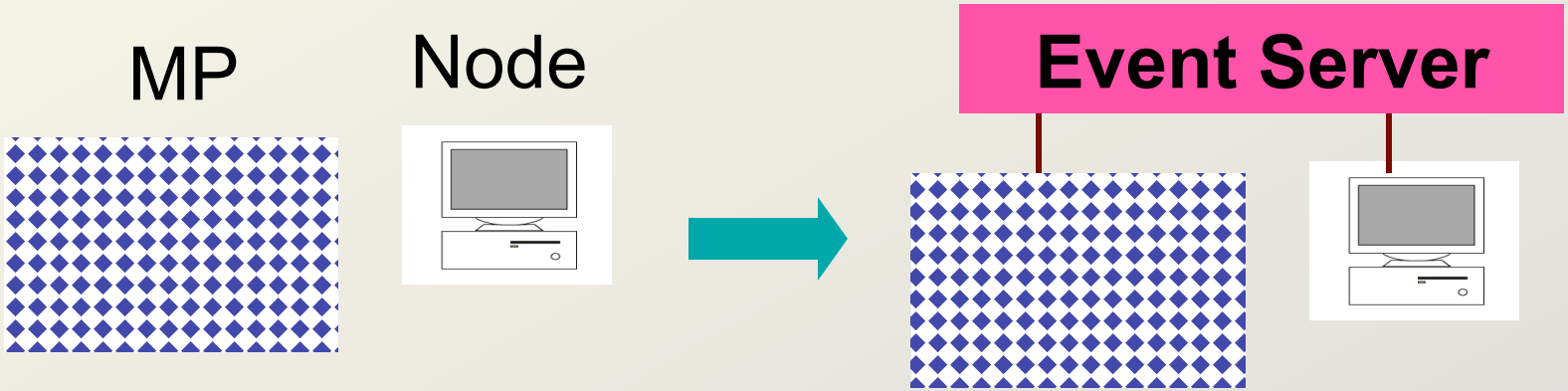
- Set **time** of event useful to others
- Some events **depend** on others- (***event chains***)
  If combination rules involve events from several tasks,
  -> a central repository with evaluation would be helpful
- Once event set, **can not be unset**

This is a **mature** project in use for years on a functioning TCV

All Change ?

No !

- Existing IPC systems (including say 1000's PCs-openMP)
  => **not** the target
- Look for solution:
  - ✓ Light-Weight
  - ✓ Limited Functionality *('Good' sense of limited…)*
  - ✓ Operates identically whether local or remote
  - ✓ Central- The whole plant can be observed



Typical things that go wrong

- Waiting an event that :
  - X **never** happens
  - X **never** returns from server
- Waiting for an event that **has happened !**
  - X **but** not while you were waiting
- Complex event sequences behave like state machines
  - X extra states from which there is no escape "**GridLock**"
- System **unresponsive** and/or does not react as "predicted"
  - X can't see present state to provide remedial actions.

- In the **absence** of the server, everything just **stops**.

Design criteria for an event/state server

Features

- **User** created events- Use “Character_Names”: **A**one, **B**two, **C**three
- Events with **dependencies** eg: ((A && B) || C)
- Event **Groups** (Run number, shot number, name stem…)
  Aone**_123**, **B**two**_123**, **C**three**_123**
- Waiting for event will **return** to client :
  after requested time (TimeOut) **or** before if event occurs during wait

Add Book-keeping

- Time (μs resolution) of **creation** and **setting** event
- **Search** events by **G**roup, **N**ame, **A**ge … (**R**egExp)
- Event **Protection** (against deletion)
- Facilities to interrogate/view complete event tree status
- Callable from **everywhere** (C, Python, Matlab, Fortran….etc.)

EPFL, Lausanne, Switzerland

More

- Scope-out simple solution: do **not** try for a “do-all”
- Make user’s external **Call-Back** possible (eg: MdsPlus events)
- **Single** thread-
  - ✓ requests fully processed with dependencies before the next
- **32/64bit** compatible.. (for arm32 etc.)
- **Many** servers on one to be machine possible
- Ensure packets of different servers can **not pollute** each-other

EVERYTHING times-out: on {server && client}
for everything
(even IP stack errors on client and/or server !)

The user process must **NEVER** hang

“DataBus” or Network ?

- **Internal** machine bus (Dbus etc.)
  (often used for communication with machine devices usb/key/mouse..)
- **Network**...
  Most net-stacks keep machine-internal communication away from the network card...
  (penalty-free when local)
- Chose **Network** !
  - ✓ Machines can talk to **themselves**
  - ✓ Machines can talk to **each-other**
  - ✓ Most machines have some **EtherNet** connection or access to a bridge
  - ✓ Routing/VPN etc. can **extend** communication

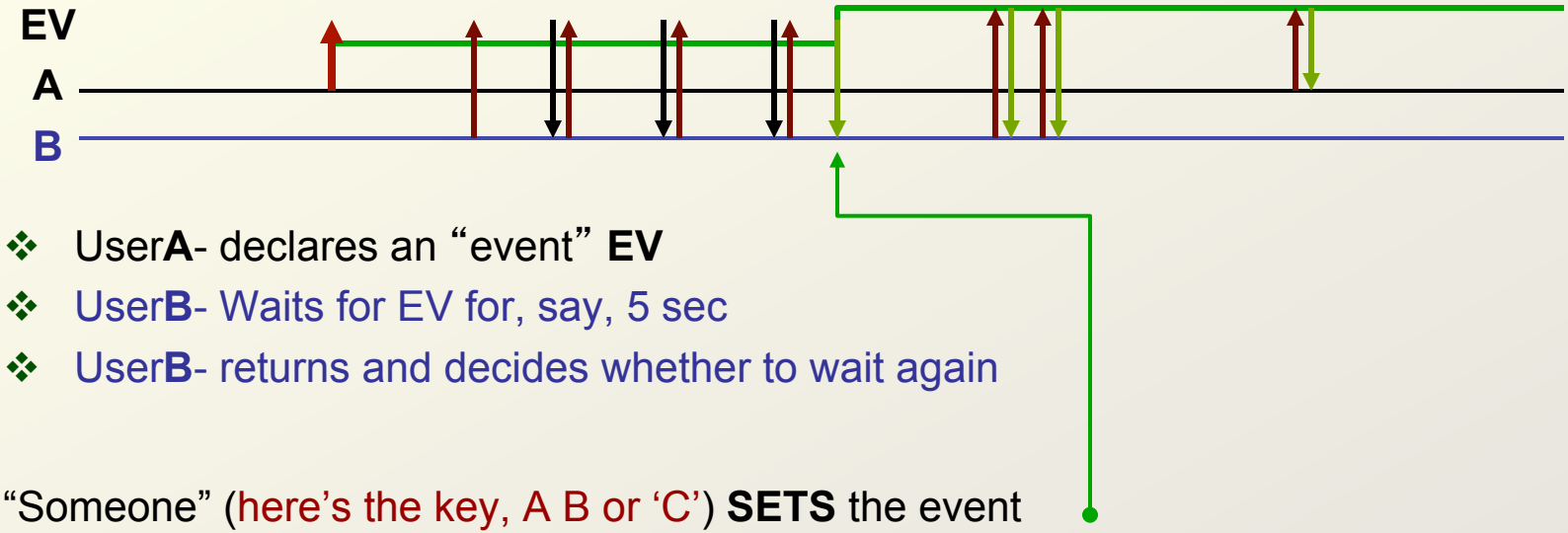
Build a first/demonstration version

System now scoped out... *Go build one*

- ✓ **Network** based
- ✓ Events declared/referenced by ASCII **name**
- ✓ **Compound** events
- ✓ **EVERYTHING** has a time-out
- ✓ **Checksum** and “magic-byte” protection
- ✓ **Single-Thread**.. i.e. NO RACE CONDITIONS or MUTEX
- ✓ **C**-language
- ✓ Allow a “user” **call-back** when event set-
  (MDSEvent implemented- either **internal** or **linked** to MDSPlus libraries)

EPFL, Lausanne, Switzerland

Target functionality



- User**A**- declares an "event" **EV**
- User**B**- Waits for EV for, say, 5 sec
- User**B**- returns and decides whether to wait again

"Someone" (here's the key, A B or 'C') **SETS** the event

- User**B** was **waiting** ?
  - ✓ wait immediately terminates: *message set returned*
- User**B** was **NOT** waiting, no problem.
  - ✓ Next time he waits for event it returns immediately
- *(event **does not exist ?** returns immediately and tells you)*

Present Demonstration Version (circa 2012 incarnation)

- Linux-based (GCC compiler)
- UDP packets (packet declared at byte-level)
- Fantastically **quick** hash table for names->pointers
- **XML** file configuration (for multiple servers, same file for clients)
- ANSI-C server: shareable library for client
- 1 Client only communicates with 1 server at a time
  (more than one server on single machine on different port)
- C, TDI, Python, Matlab, MDSip(tdi) glue
- Select/Delete/Search events by: **name**, **shot#**, RegExp, **age**…

- Can define an MdsEvent to be set upon an event going True
  (generates an internal UDP-MdsEvent or be linked to MdsPlus libraries)**- One-Shot, of course**
- **Long** event names available *(thanks to xsl)*
- Compiled on X86, ARM, PowerPC
- Ran for weeks->months with xxx thousands of creations/deletions *(without crashing or hogging RAM…)*

Example_I (100 is a **serverID**) SimpleEvent

- ev**New**(100,"Event1")
- ev**Get** (100,"Event1") *returns* ***"evFalse"***
- ev**Wait**(100,"Event1",**10.2**) *comes back after **10.2s*** ***"evTimeout"***

- ev**Set** (100,"Event1")

Now

- ev**Wait**(100,"Event1",10.2) *returns immediately* ***"evTrue"***

- Client **decides**: "wait again" or "try later"

Example_II (100 is a serverID) CompoundEvent

- evNew(100,"**E1**");evNew(100,"**E2**");evNew(100,"**E3**"); evNew(100,"**E4**");
- (logical operators '**&**' '**|**' '**^**' available)
  eg: (E1 & E2) | (E3 &E4) is written "E1,E2,E3,E4", "01&23&|"

- evNew(100,"**Cev**","**E1,E2,E3**","**01&2|**");
  [here: **Cev** is a '**compound**' event (RPN logic)]
  i.e. set either by setting (**E3**) **or** (**E1&&E2**)

- evGet(100,"**Cev**") *returns* "**evFalse**"
- evSet(100,"**E1**");
- evGet(100,"**E1**") *returns* "**evTrue**"
- evGet(100,"**Cev**") *STILL returns* "**evFalse**"
- evSet(100,"**E3**")
- evGet(100,"**Cev**") ***NOW** returns* "**evTrue**"

Sordid details…

- If an event is deleted ?
  - ✓ It's **deleted**
- A compound event is deleted ?
  - ✓ It's **deleted**
- Deleted event in use by a compound event ?
  (eg: event E1 is deleted in Cev)
  - ✓ Make E1 forever **TRUE** in calculation of Cev

WHY ?

- 'cause someone **has** to **decide**:
  deleting an event should not cause other events to become non-defined and/or hanging

Possible future additions

- Exchange UDP for more distributed **I/O**
  eg: ZeroMQ, RabbitMQ
  (This can handle ssh, tcp/ip, re-ordering etc.)

- **Protect** events more (lock certain events to clientID ?)

- **Propagate** and/or **Duplicate** Events between servers
  (fall-back-servers, drone-servers)
- **Firewall** for outside syncs/monitors

- Event-**Groups** (for a particular activity eg: shot)
  eg: create
  init_123, start_123, pray_123, trig_123, stop_123 in one call etc.

Possible Uses

- Organise a heterogeneous **workflow**
  - ✓ (after TCV-shot, sequence Shot analysis, etc.)
- **Track** a running process/processes
  - ✓ (each stage can set a flag that is see remotely)
- Synchronise a **distributed** set of computers
  - ✓ (acquisition, control etc.)

- **Lightweight** Inter-Process-Communication (IPC)

See "**Data**" section

- Propagate: characterString, Int-array, float-array, double-array

Security Issues- still basic

Packets have **3** security levels:

1. Packets **sequentially** numbered
   - ✓ *(Refuse a reply to a question with a backwards sequence number)*
2. Packets have a different "**magic**" number /server and 32-bit **checksum**
   - ✓ (Refuse a packet (either-end) with the wrong "magic/checksum")
3. Client **pid** sent and recorded in server-
   - ✓ (Refuse a reply that does not contain the requestor's PID)

- ✓ Ensure that reply is to the **correct question** and nobody has "inserted" a "replay" packet back into system
- ✓ (and, less likely, that there was an **error** in the communication since there is always the IP-checksum and a packet **checksum**)

Performance ?

- Even “banal” geode device (€100) handles over 1000 requests/sec
  - ✓ (34 bytes each, no sweat)
- Employed an efficient “**hash-table**” to translate the event names into a pointer to the event data (from Australian PhD)
  - ✓ (Handles many 10’000/sec on simple hardware)
- If “**Compound**” events (requiring calculation) are not over numerous, only results in **acceptable** performance hit.
  - ✓ (More than 1000 re-calculations/second are possible on simple hardware)

- All code in ANSI-C. Should compile on more or less **anything**
- UDP packets are easy to construct **externally**:
  (native Python/Perl/TDI or anything else, is not problematic)

Example XSD to HTML result

- CLI- all event **metadata** to terminal available
- Other "**snooping**" and "**status**" commands available

- Server Checkpoint->file created upon demand (XML)
- **evtGeode.xsd** translates checkpoint to HTML
- *Other "filters" can be used to extract data to programs (eg: rexExp) and/or create live "html" pages to follow some selected parameter(s)*

Events List

Position	ID	Name	state	flag	shot	cTime	sTime	evDep	evList	evFunc	evLogic
1	167925720	bpd_event2	evOff	0	0	Wed Mar 21 14:52:04 2012		167926600 - bpd_revent			
2	167926160	bpd_event3	evOff	0	0	Wed Mar 21 14:52:04 2012		167926600 - bpd_revent			
3	168022112	Bdata1	evInt	4	0	Wed Mar 21 14:52:09 2012	Wed Mar 21 14:52:13 2012				
4	168022520	BdataS	evChar	4	0	Wed Mar 21 14:53:14 2012	Wed Mar 21 14:53:31 2012				
5	167925280	bpd_event	evOff	0	0	Wed Mar 21 14:52:04 2012		167926600 - bpd_revent			
6	167926600	bpd_revent	evOff	0	0	Wed Mar 21 14:52:04 2012			167925280 - bpd_event 167925720 - bpd_event2 167926160 - bpd_event3	01&2l	bpd_event,bpd_event2,bpd_event3

Example: for TCV-shot

In **shot-prepare** phase (say shot #12345)

- **Create** tcvStart_12345, tcvAcquire_12345, tcvEnd_12345…
- **Create** diagnostic compound events
  - Thomson_12345, NPA_12345, Toray_12345…
- **Run tasks**: Thomson, NPA, Toray… that were already (or not) waiting
  (make events ALSO return if tcvAbort_12345 is set)

After **tcvAcquire_12345**

- **set** Events for **Diag** and **Calc** in order…

After **tcvEnd_12345**

- events with shot 12345 are **deleted**:
  _names = evStatI_del(100, 12345, _states);
  (returns deleted names and their states before deletion)

Résumé

- **Robust** UDP-message Event and State (and Data) server
  - ✓ callable from many machines and different PIDs
- Return to user after a specified time so user does **not** "lose" control
- Server can easily handle **1000's** of clients-
  - ✓ *(you can run many servers on same/different machine too….)*
- **Supervisor** and **logging** routines aim to help debug sequencing errors
  - ✓ (understand surprises)
- Only **3** main calls: **New**, **Set**, **Get** (ok... 4, **Delete**…)
- Implemented in ANSI **C** (shareable/static libraries)
  - C, Fortran and glue routines for TDI, Matlab-
  - Python "event" class available
  - *Easy port to PERL/PHP/R etc.*
- "**Protection**" could be enhanced as needed

User Commands (user API)

- int evt_ldbg(int deb); [change local debug level]
- int waitEventP(int server, char *name, float *waitP);[wait event for a time]
- int newEvent(int server, char *name, char *logicS, char *logic); [New Event]
- int getEvent(int server, char *name); [Get state of event]
- int statEvent(int server); [Remote list of all events]
- evRec *CstatEventR(int server, char *name); [Get a complete event record]
- int statEventR(int server, char *name); [Get complete event record]
- int statEventL(int server, char *pattern, char *rep, int replen, char *logic);
  [get list of events matching a REGEX pattern]
- int statEventI(int server, int ival, char *rep, int replen, char *logic); [List with Shot i]
- int statEventT(int server, int ival, char *rep, int replen, char *logic); [List older ival sec]
- int statEventLD(int server, char *pattern, char *rep, int replen, char *logic); [&& delete]
- int statEventID(int server, int ival, char *rep, int replen, char *logic); [&& delete]
- int statEventTD(int server, int ival, char *rep, int replen, char *logic); [&& delete]
- int setEvent(int server, char *name); [Set event to True ONLY if False]
- int rXMLEvent(int server, char *name); [Write a list of all events to file name]
- int printEvent(evRec *evP, char *name); [Local function to print an event to screen]
- int listEvent(int server, char *names, char *logic); [Get event Dependencies]

Data Versions

- int getEventData(int server, char *name, void *dat, int *len); [Get DataRecord]
- int newEventData(int server, char *name, int type); [New DataRecord]
- int setEventData(int server, char *name, int type, void *dat, int len); [Set DataRecord]

DataMode

Since events have reserved memory on server, a simple extension allows the user to record and recover (read) the following:

- Up to n-characters (n=350 in present incarnation)
- Up to n/4 integers
- Up to n/4 floats
- Up to n/8 doubles

Although these are NOT “events”, they use the same communication and server and record the time at which the data changed.

This “data” extension used to share limited amount of data between tasks- or it could be an “expression” ho ho ho

DataMode API *(tdi example)*

- evNewDat(100,”Bdata_I”,2) *(1:char,2:int,3:float,4:double)*
- evSetDat(100 ,”Bdata_I”,[11,22,33,44]);

- evGetDat(100 ,”Bdata_I”)
  returns [11,22,33,44];

- evStatE(100,”Bdata_I”) *gives information on last write*
  Bdata_I->[Bdata_I] Flg[4] Sht[0]
  cre[Wed Mar 21 11:22:53 2012] set[Wed Mar 21 11:22:59 2012]
  evInt 33,55,66

Auxiliary API routines *(tdi example)*

ALL Details of an event are user-accessible with API:

```
evStatDet(100,'BdataI',_Rname,_timeCreate,_timeSet,_state,_evShot,_flag,_len,
    _evDep,_logic,_evList,_evFunc, _evData);
```

From here all the record metadata is accessible:

```
evTimeStr(_timeCreate)          -> "Thu Mar 22 09:15:31 2012“
evTimeStamp(_timeCreate) )      -> 29731.6 (sec since 0:0)
```

XML configuration file

Server/client settings

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="serialGeode.xsl"?>
<doc xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="serialGeode.xsd">
  <dbg>1</dbg>
  <timeout>30</timeout>
  <hashsize>11000</hashsize>
  <magic>0X3456</magic>
  <net>eth0</net>
<!-- name a remote server with a user and control port -->
  <node id="100">
  <hostname>crppmac70</hostname>
  <remoteport>300</remoteport>
  <controlport>301</controlport>
  <control>0</control>
 </node>
  <node id="101">
  <hostname>crppemb01</hostname>
  <remoteport>300</remoteport>
  <controlport>301</controlport>
  <control>0</control>
 </node>
</doc>
```

XML checkpoint file (event example)

```
<event id="167926160">
  <name>bpd_event3</name>
  <flag>0</flag>
  <shot>0</shot>
  <cTime>
    <string>Wed Mar 21 14:52:04 2012</string>
    <sec>1332337924</sec>
    <usec>46001</usec>
  </cTime>
  <sTime>
    <string></string>
    <sec>0</sec>
    <usec>0</usec>
   </sTime>
  <state>evOff</state>
  <flag>0</flag>
  <evDep><eventd idd="167926600">
    <name>bpd_revent</name>
   </eventd>
  </evDep>
  <evList></evList>
  <evFunc></evFunc>
  <evLogic></evLogic>
  <evData>[]</evData>
</event>
```

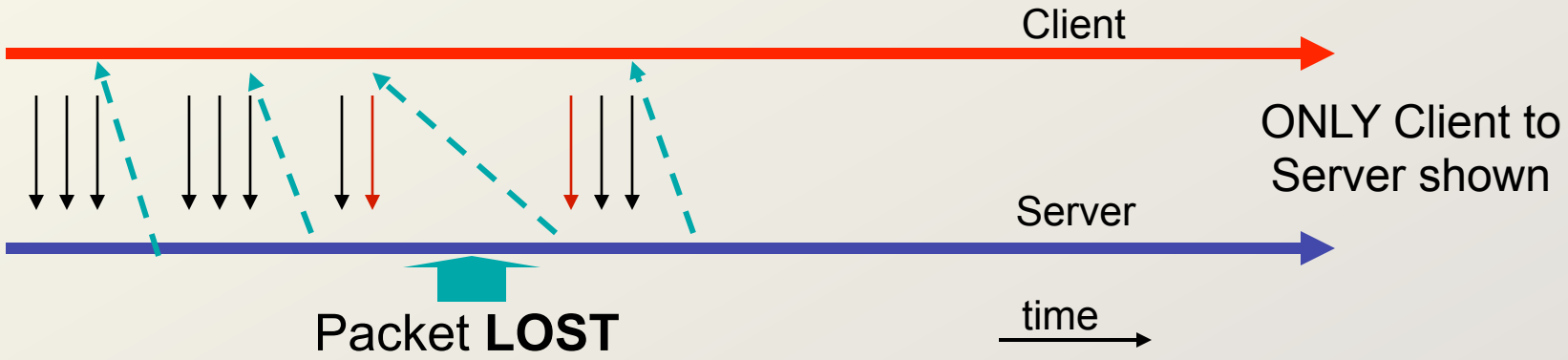
XML checkpoint file (Data example)

```
<event id="168022112">
 <name>BdataI</name>
  <flag>4</flag>
  <shot>0</shot>
  <cTime>
    <string>Wed Mar 21 14:52:09 2012</string>
    <sec>1332337929</sec>
    <usec>811263</usec>
  </cTime>
  <sTime>
    <string>Wed Mar 21 14:52:13 2012</string>
    <sec>1332337933</sec>
    <usec>251412</usec>
  </sTime>
  <state>evInt</state>
  <flag>4</flag>
  <evDep></evDep>
  <evList></evList>
  <evFunc></evFunc>
  <evLogic></evLogic>
  <evData>[33,55,66]</evData>
</event>
```

Brief Aside: Communication “TCP-like”

We “count upon” **flawless** communication between devices using the **TCP/IP** protocol- (SSH, FTP, Telnet, NFS…)

- Socket_pair is set up between **client** and **server**:
  error correction and packet re-ordering ensure data streams are synced and lost packets **re**-transmitted.
- This creates **work** for client and server that takes time.
- Reliable by **design**- Lost packets are noticed and resent



Alternative “UDP-like”

FAST

- **Little** implicit work for kernel
- Server **waits** for requests on buffered port
- Client **sends** a message that may be **lost** (*or discarded*)
- Client can **send** another command *(confirm reception)*

Furthermore

- **No socket-pair**: Client sends packet
  can (**or not**) await a reply on that socket before closing
- Many 1000/s of packets handled by **modest** system
- Packets “cheap”: you can afford to send **many**
- For closed environment (eg: inYourLab)
  packet loss/error is **negligible** in any case.

HOWEVER: Guard against network errors ASAP

UDP non-error handling

- ✓ Requests **without** reply: repeated by client or forgotten
- ✓ **Fast** turn-around time
- ✓ Protocol **standard** & **trivial** for external implementation
  (backward compatibility where possible)
- ✓ Add **checksum** (in-packet) for error immunity