THE TINE CONTROL SYSTEM PROTOCOL

HOW TO ACHIEVE
HIGH SCALABILITY
AND PERFORMANCE

Data Flow and Scalability

- Most Scalability issues in a distributed system are concerned with *data flow*.
- Most of the data flow in a distributed control system is concerned with *readback* data.
  - Display
  - Middle Layer logic (automation decisions)
  - Archiving
- The *Control System Protocol* is concerned with data flow.
  - Server i/o -> clients
  - Load on server
  - Load on network
  - API

Control System Protocols

- Initial Driving Force :
  - KISS:
    - e.g. transaction polling is easy to program!
- Initial Consequence:
  -
    - Fewer options often lead to more and bigger problems
    - Solutions sometimes artificial
    - Corollary: And later on with more tools available:
      - Still use the hammer (‘cause that’s what you know)!

Data Flow Memes : 0th Order

- Transaction-based Client-Server
  - Client asks, server responds
  - KISS (no management tables) !
  - Suitable for *small* systems
  - Server Load:

$$L_S \sim N_C \times N_T \times L_D \times U_T$$

L_S = Load on Server/sec (CPU cycles spent)
N_C = Ave. num clients
N_T = Ave num transactions / client
L_D = Ave Load handling a dispatch
U_T = Ave update rate

Threads ?

Multi-Core ?

Transaction-based Client-Server

Data Flow Memes : 0th Order

- Network Load:

$$L_N \sim N_C \times N_T \times P_T \times U_T \times 2$$

L_N = Load on Network (bytes/sec)
N_C = Ave. num clients
N_T = Ave num transactions / client
P_T = Ave Transaction Payload
U_T = Ave update rate
2 = outgoing + incoming payloads ~equal

Threads ?

Multi-Core ?

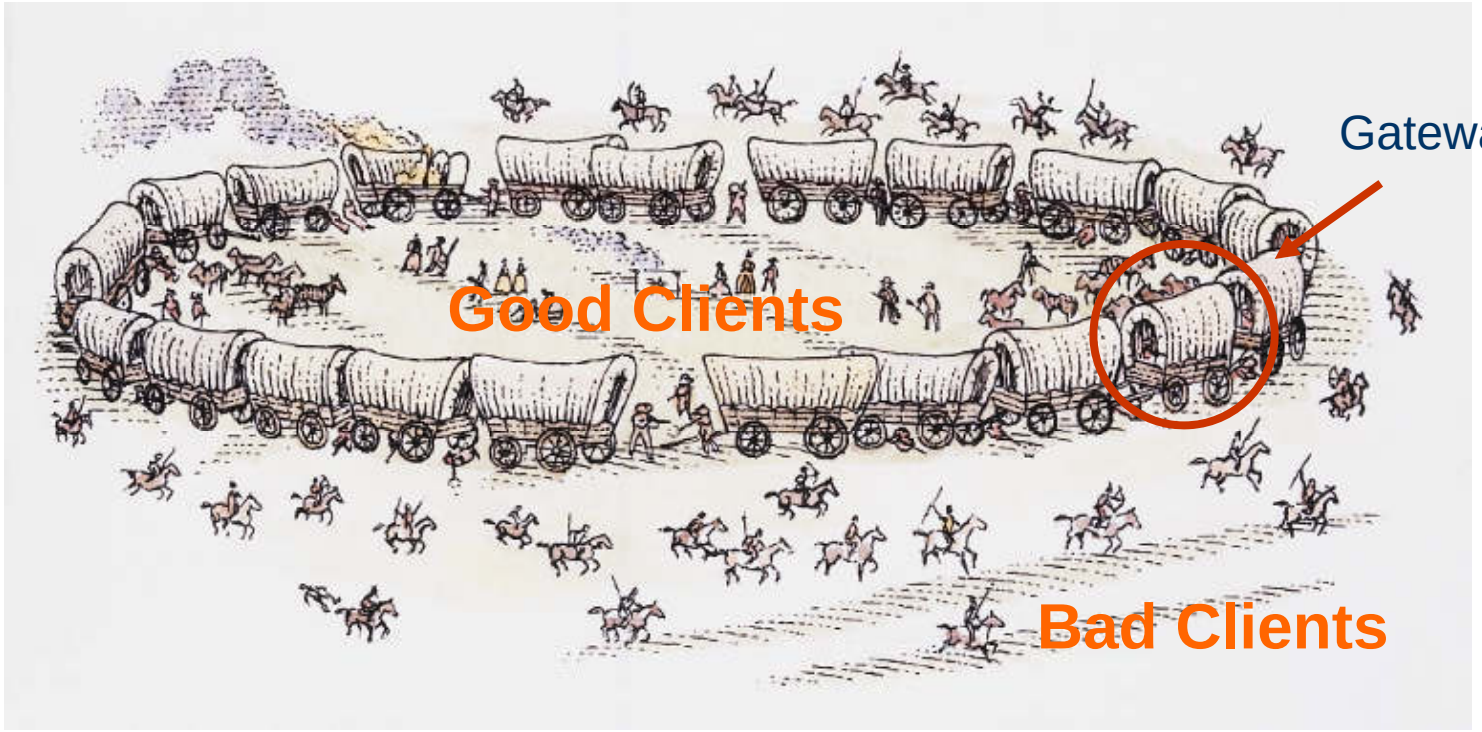
- Increase Scalability => Reduce the Load

Reduce any of these factors !

Transaction-based Client-Server

Data Flow Memes : 0th Order

Possible solutions to Scalability Problems :



Data Flow Memes : 1st Order

- Contract-based Publish-Subscribe
  - Kiss **KISS** goodby !
    - Contract and connection management
    - Transaction => managed contract + table of clients
    - Larger systems
    - Server Load :

$$L_S \sim N_T \times L_D \times U_T$$

L_S = Load on Server/sec
N_T = Ave num transactions / client
L_D = Ave Load handling a dispatch
U_T = Ave update rate

N_C (Number of Clients) no longer a factor !

Data Flow Memes : 1st Order

- Network Load:
  - Similar, but:
    - Factor 2 gone!
    - Can use '*Send on Change*' to reduce U_T
    - Can use *multicast* to reduce N_C
- Great Benefit to Scalability!
  - BUT: API can still allow inefficiency !
  - AND: Who is doing the programming ?

$$L_N \sim N_C \times N_T \times P_T \times U_T$$

L_N = Load on Network (bytes/sec)
N_C = Ave. num clients
N_T = Ave num transactions / client
P_T = Ave Transaction Payload
U_T = Ave update rate

APIs and ‘Joe, the Programmer’

- Some APIs still allow or even ‘only allow’ synchronous polling !
  - Easy for Joe to understand
    - no callbacks or events …
  - N_C once again a factor in the server load!
- Some APIs guide Joe to use individual transactions for all readback values
  - Each of the 300 BPM positions, status, etc.
  - Each of the 600 Ion Pump pressures, status, etc.
  - Each of the 1000 PSC currents, setpoints, status, etc.
  - N_T can be very large !
- But: These client applications are driving the data flow !

Contract-Coercion

Data Flow Memes : 2nd Order

- Publish-Subscribe with Contract-Coercion
  - Focus on ways to reduce N_C and N_T in server load and network load
    - Police action ?
      - Find Joe and tell him to do it differently.
    - Push the data ?
      - multicast everything!
      - Network load could increase dramatically
      - Client load could increase dramatically
  - Goal: Keep the APIs and let Joe do what he wants.

Contract-Coercion

Data Flow Memes : 2nd Order

- Analyze the transaction request
  - Map to an existing contract if possible
  - Anticipate future requests and renegotiate the contract with the client
    - e.g. “if he’s asking for BPM#1, then he’ll probably want BPM#2 as well”
  - Guide synchronous and asynchronous acquisitions
    - Don’t monitor ‘static data’
    - Don’t synchronously poll monitorable data.
    - Trap ‘foolish’ update intervals
  - KISS is a distant memory

  - Briefly review 3 Control System Architecture Models …

Control System Models (a review)

- Model I: Database Model
  - EPICS, VISTA (i.e. VSystem not the OS)
  - Control system data are *elements in a database*.
  - Transfer *Process Variables*
    - pvData have names
    - Actions are ‘get’, ‘set’, ‘monitor’
  - BUT: Some things aren’t variables at all !
    - e.g. command and calls

Control System Models (a review)

- Model II: Device Server Model
  - TANGO, DOOCS, ACS, STARS*, TINE*
  - Elements are controllable objects managed by a device server.
  - Instance of such an object is a *device*, with a hierarchical name.
  - Actions pertaining to a device given by its *properties* !
    - i.e. *get*, *set*, *monitor*, *call* some *property* OR *command*
  - BUT: some things aren't devices !
    - e.g. "*" is NOT a device.
  - AND: some services are *Property-orientated* !

Control System Models (a review)

- Model III: Property Server Model
  - STARS*, TINE* (maybe ACS?)
  - Elements are *services* with *properties* (or methods)
  - Same basic hierarchy as Device Server Model
  - Properties have Keywords
    - (instead of Devices having Properties)
  - e.g. Middle layer services
    - Name Server
    - Central Alarm Server
    - Central Archive Server
    - CDI Server
    - etc.
  - BUT: Not everything divides cleanly into *Device Server* or *Property Server* !

Transaction Coercion

- Steps to reduce N_C, N_T, L_D, and U_T
- Refer to the *Property* in the above models
  - the ‘get’ and ‘monitor’ operations produce most of the data flow in a distributed control system.
- Some Examples

Transaction Coercion

- Multi-Channel Arrays (**MCA**)
  - An array of all values of all devices for a given *Property* with a well-known order.
    - Same units, settings, etc.
  - Registered Property can declare itself to be an MCA.
  - Strict OO Device Servers can declare device *groups* with MCA characteristics.
  - e.g. BPMs, BLMs, IonPumps, Temps, PSCs, etc.
  - But Joe's Client Panel will have a widget for a each thing individually !
  - What to do?

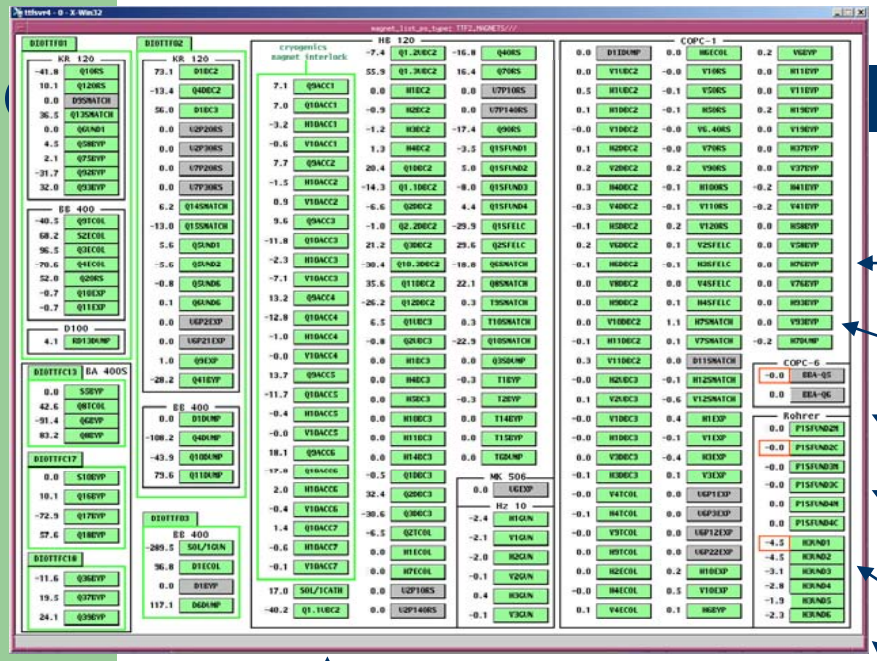
Transaction Coercion

- Contract-renegotiation
  - delivers entire MCA
  - informs client that widget #18 needs element number #43 of the array !
- N_T is decimated !

FLASH DDD PS Panel with TINE Multichannel Polling

10 connections/property (was ~260)

50 contracts (was > 1000)



STEERER/SHGROUP (57)

STEERER/SVGROUP (48)

QUAD/QDGROUP (40)

DIPOLE/DIGROUP (19)

UNDULATOR/UNGROUP (7)

MAIN/MNGROUP (6)

SEXT/SXGROUP (3)

SOL/1GUN (1)

SUN (multi-server)

TTMAG-AB/GROUP

(PC104, 41 elements)

TTMAG-CD/GROUP

(PC104, 41 elements)

PETRA BPMs

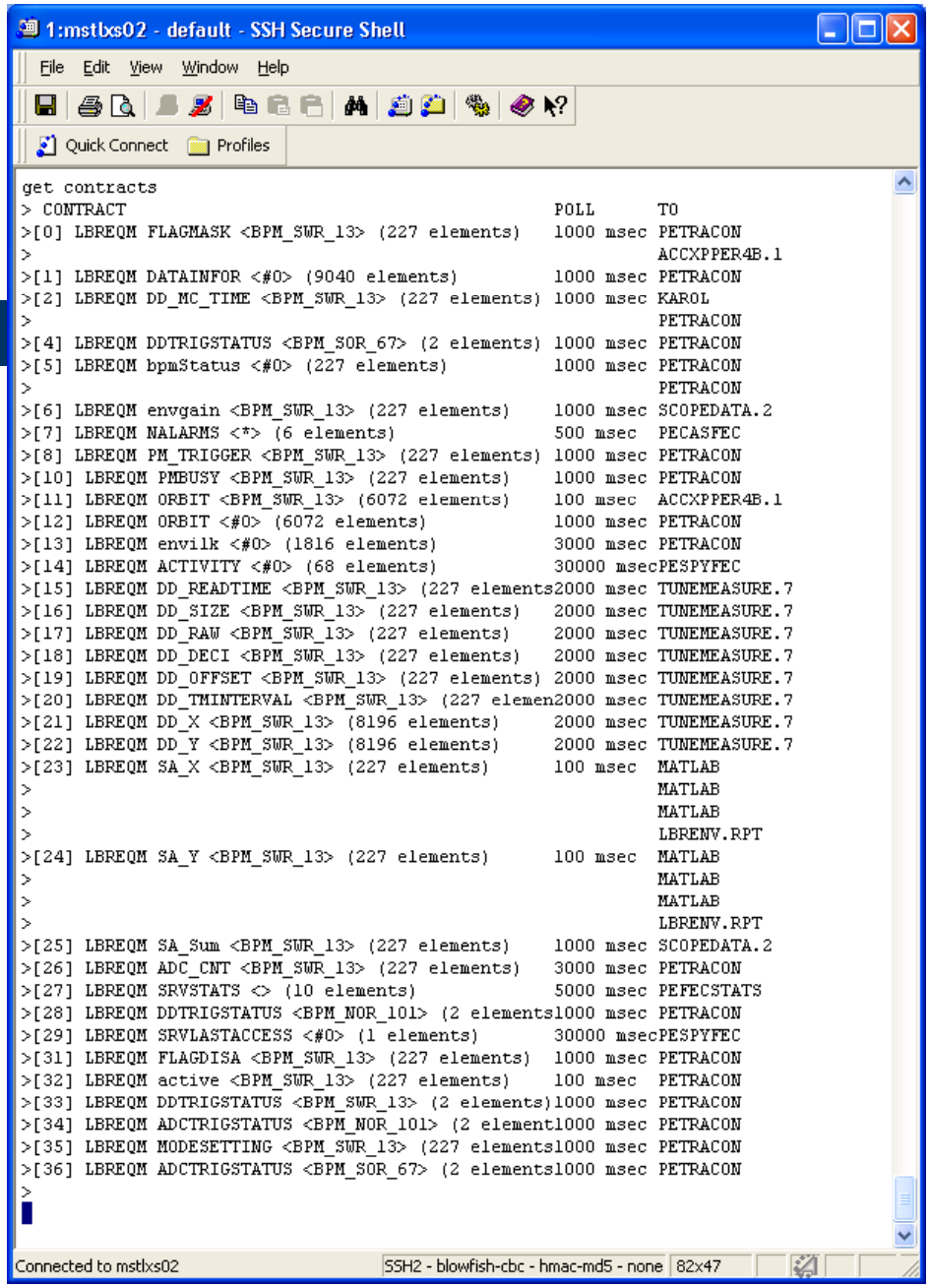
227 Libera modules

~ 15 Client

~ 35 Contracts

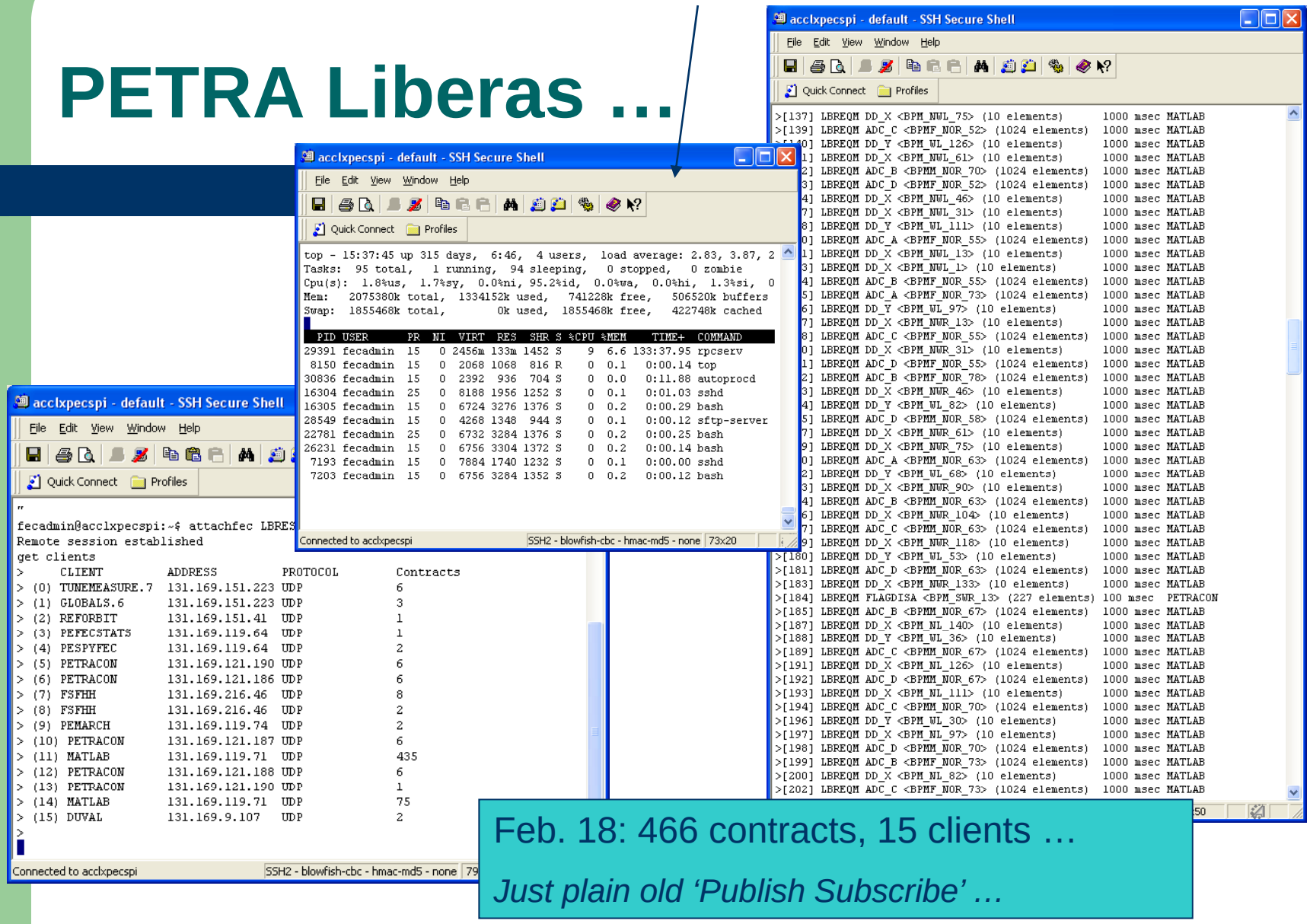
1 run-of-the-mill PC with Ubuntu Linux

~ 7% CPU load !



PETRA Liberas ...

CPU: 9 %

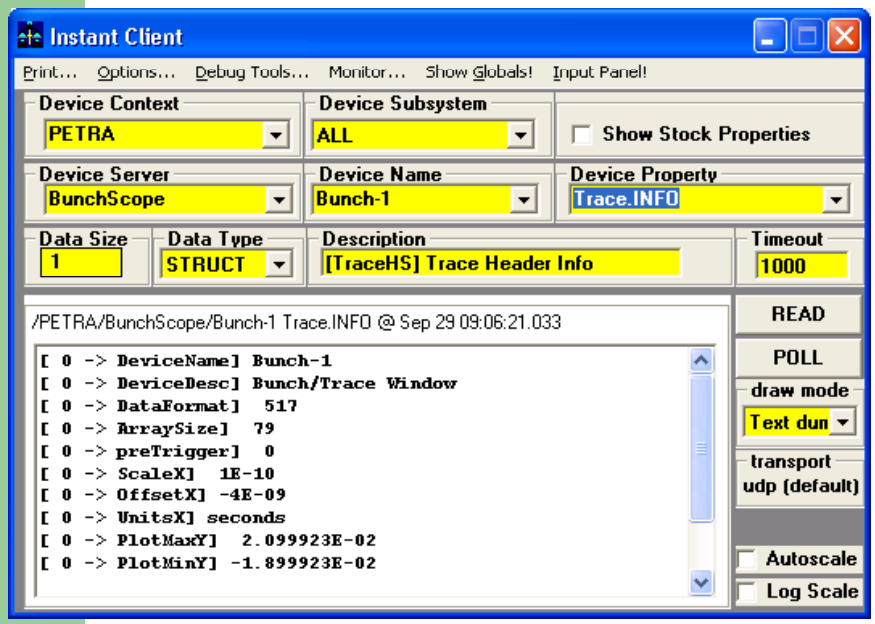


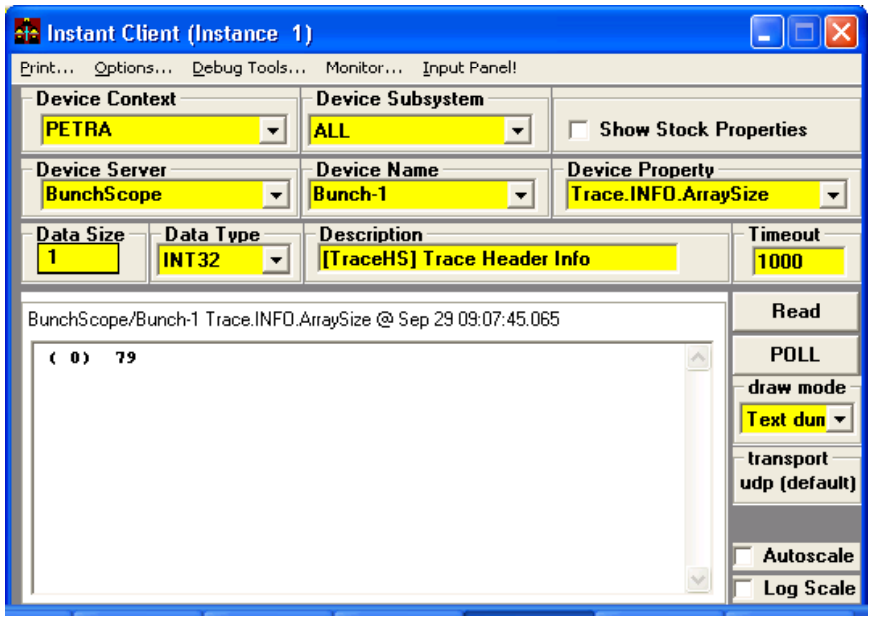
Feb. 18: 466 contracts, 15 clients …

Just plain old 'Publish Subscribe' …

Transaction Coercion

- User-defined Structures
  - Requests for individual fields deliver the entire structure !
  - Reduce N_T !





Transaction Coercion

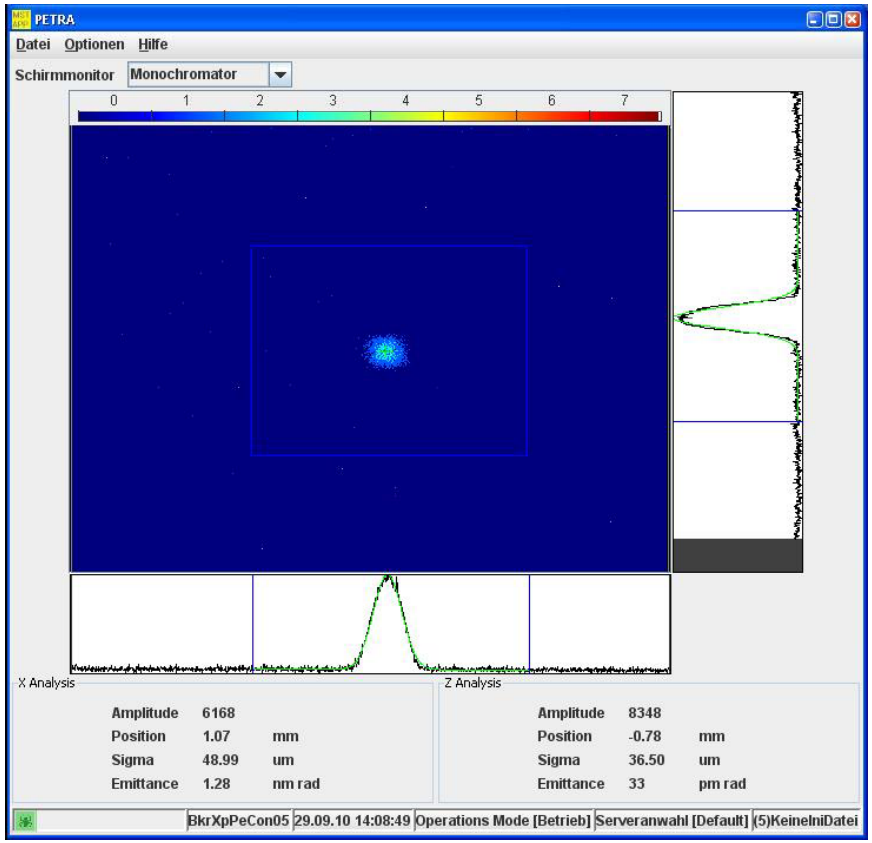
- Polling Intervals and Scheduling
  - Eager Clients:
    - Need to know something '*as soon as it happens*'
    - Sometimes use ridiculously high update rates.
  - Server can enforce a '*minimum polling interval*'!
    - Client is informed and makes an adjustment
  - Server '*schedules*' the data when it changes.
    - e.g. DESY2
      - cycles @ 6.25 Hz
      - clients attempt to update @ 10 or 20 Hz
      - Server establishes minimum polling rate @ 1 Hz but schedules requested data @ the 6.25 Hz hardware trigger rate
      - clients now really do get the data '*as soon as it happens*' !
  - Reduce U_T !

Transaction Coercion

- Steering the Acquisition Mode
  - Properties can reject synchronous calls
    - Client can react and insert an asynchronous listener under synchronous looking API calls.
  - Properties can reject monitors
    - Client is informed that monitored data are static
  - Properties with large Payload can enforce multicast acquisition !
    - Monitors coerced into listening for multicasts.
    - Synchronous calls rejected.
  - Reduce N_T and N_C !

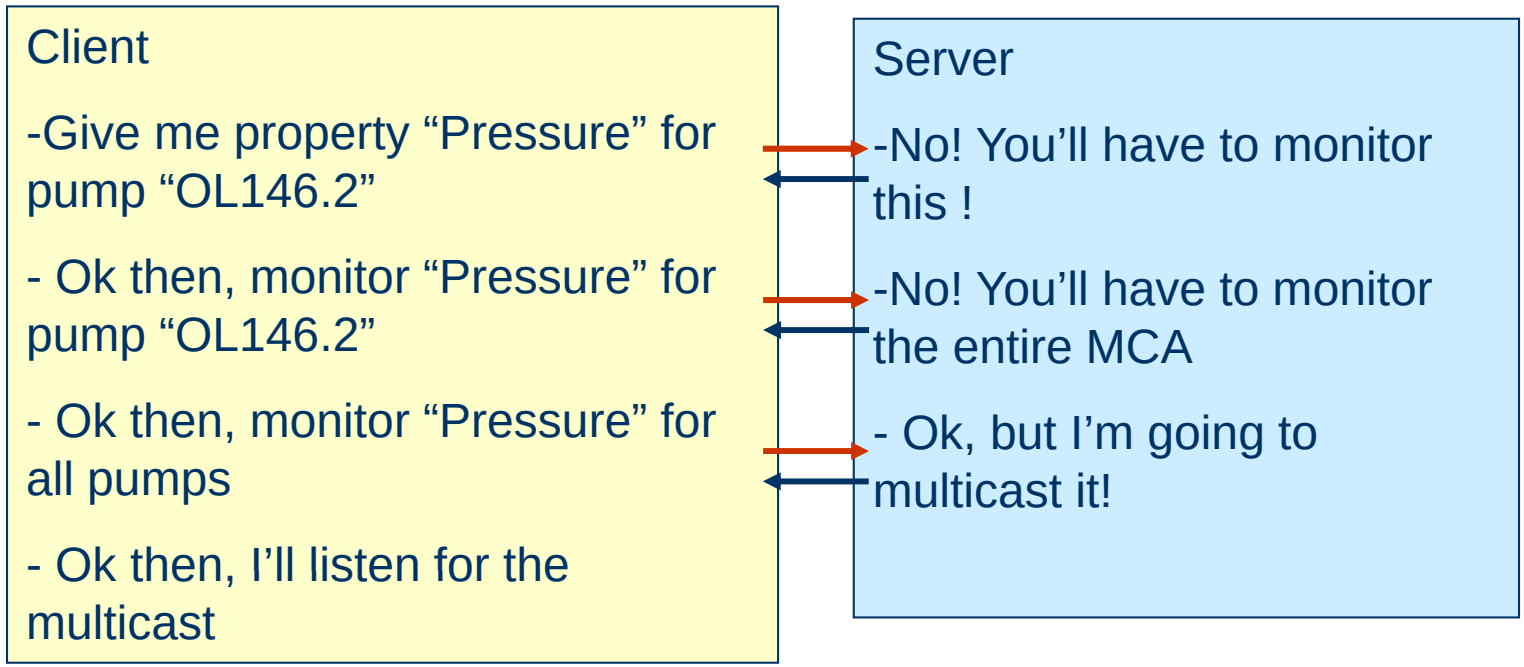
Transaction Coercion

- PETRA3 Video Example:
  - Mixed Gbit/100-Mbit Net
  - Large images (large P_T)
- Tune the transport Parameters !
  - Enforce multicast
  - Minimum polling interval
  - Redirect non-Scheduled Property to Scheduled Property
- $N_T = 1$, $N_C = 1$



A Server takes control of its Clients

Example: doing 1 thing for 1 effective client instead of 600 things for 10



Conclusion

- New Data Flow meme: Contract-Coercion
  - Can provide high scalability (without API restrictions)
- Yes, you can …
  - Run epics2tine
  - Run tango2tine
  - Use doocs (which has TINE embedded)
  - Use stars which has a TINE bridge

- http://tine.desy.de
- Email: tine@desy.de