

Spearfish: Real-time Java-Based Underwater Tracking of Large Numbers of Targets

Robert A. Cross, Ph.D.

Functional Team Leader:
Underwater Tracking and Display Systems

Naval Undersea Warfare Center Division, Newport

DISTRIBUTION STATEMENT "A": Approved for Public Release; distribution is unlimited

Summary

- Spearfish is 100% Java: Windows + Linux
- Deployed and in use right now at Navy ranges
- It tracks to fine-grained accuracy at depth
- It tracks large numbers of targets
- It can post-process a data set quickly
 - 100x real time on commodity hardware
- We eat our own dog food
 - This is software that we take to sea

Missions Supported

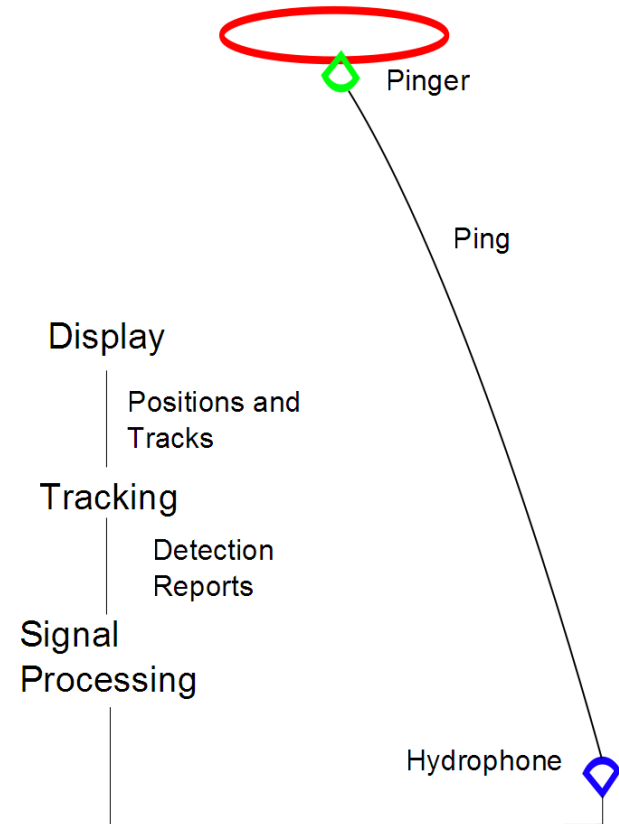
- Training (e.g., “war games”): post-exercise tactical analysis and range safety
- Test and evaluation: absolute and relative accuracy critical
- Novel systems: reconfigure existing capabilities to support new requirements

Problems

- The ocean is deep and dark
- The ocean is great at absorbing energy
- The ocean makes plenty of noise already
- The test or exercise is designed to meet operational requirements
 - Not tracking convenience

Range Systems Overview

- Pingers emit
- Hydrophones listen
- Signal processors detect
- Tracking localizes and tracks
- Spearfish:
Underwater Tracking and Display



Glossary

- Ping: An encoded signal carrying a data payload
- Pinger: Transmits pings
- Sound speed profile: The speed of sound at depth
 - On the order of 1500 m/s
- Hydrophone: An underwater microphone
- Detection report: a data packet from the signal processor
 - Ping X arrived at hydrophone Y at time T

High-level Goals

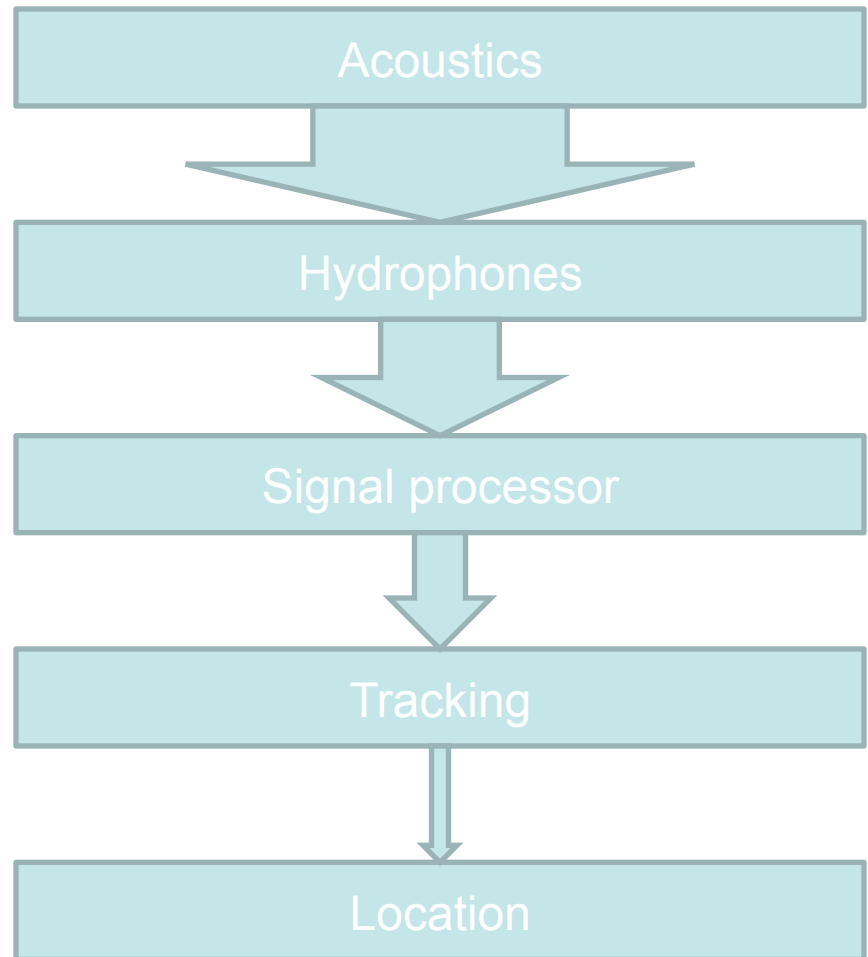
- Range safety
- Detect and track range participants
- Graceful scaling and degradation
- Alert on error conditions
- Accuracy: relative and absolute
- Real-time

Real-time? Multiple types

- Timing: accuracy and precision
 - Sub-millisecond time-tagging (signal processors)
 - Track accuracy is bound by both
- Latency
 - Dominated by transmission through the water
- Interface
 - Multithreaded data-flow architecture
 - Don't delay the processing to update the screen (and vice-versa)

High Level Tracking Components

- Initial configuration
 - Hydrophone locations
 - Environmental conditions
- Input
 - Detection reports from signal processing
- Output
 - Target location at a particular time



Components: Pingers

- A ping is an encoded acoustic signal
- Data payload: target ID, ping sequence id and, sometimes, depth
- A pinger emits pings on a specific frequency at a fixed repetition rate
- Directional bias: dependent on construction and installation

Components: Splash

- A splash is anything that is not a ping
- Examples:
 - Broadband impacts
 - Mechanical transients
 - Active emissions
 - Mammals

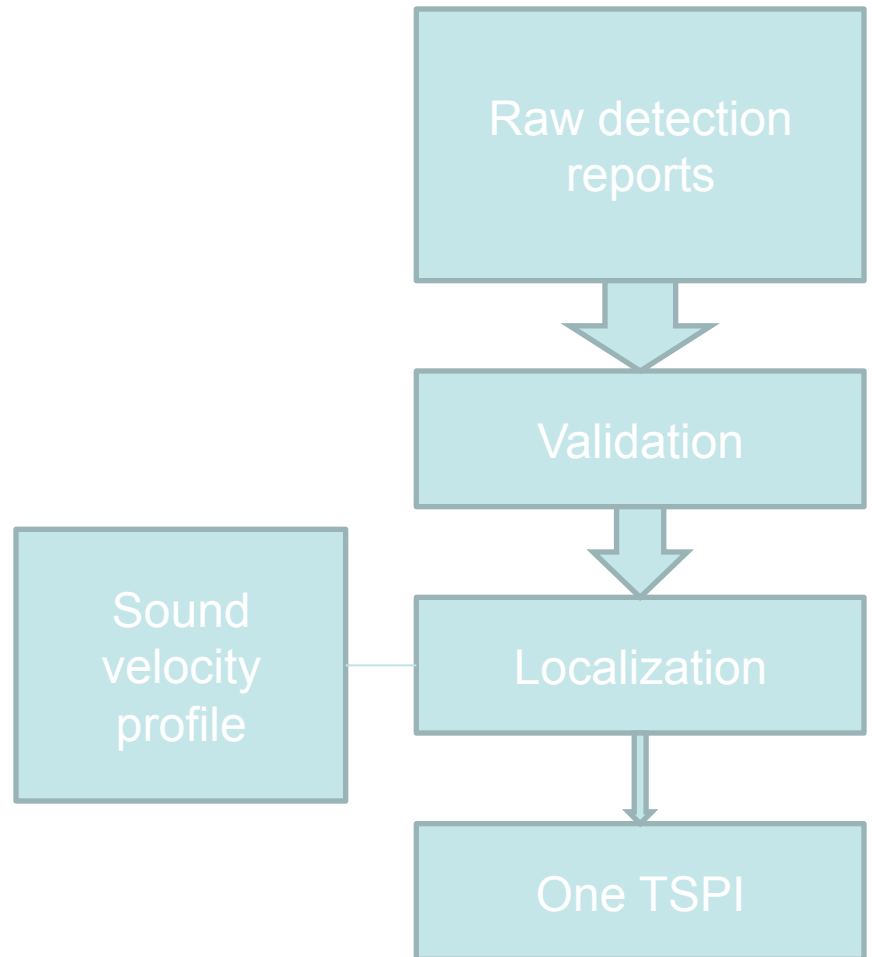


Components: Hydrophones and Signal Processing

- Hydrophone detects sound
- Signal processing receives voltage
- Converts ping (or splash) sound to detection report
 - Ping data payload + time of arrival at hydrophone
- Limitations
 - Noise in water => corruption or loss of data
 - False alarm rate => spurious detections
 - Bad angles, long range => reduced signal => loss of data

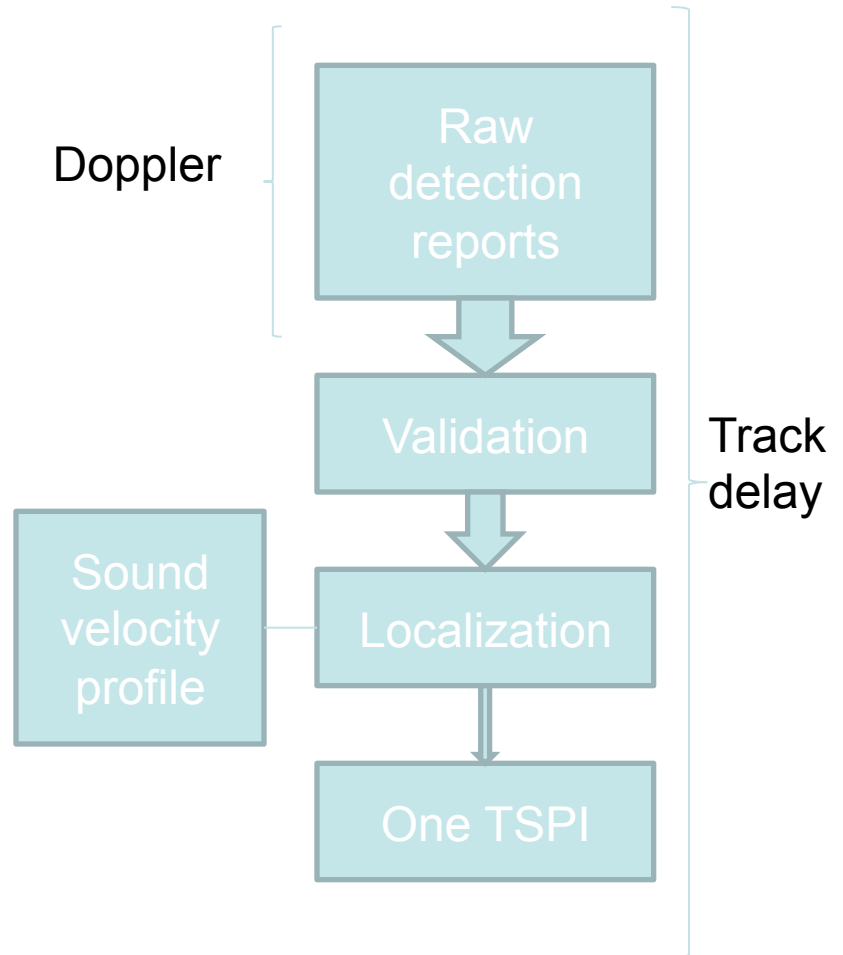
Components: Tracking

- Tracking receives raw data
- Validation: sifts out the valid reports
- Localization: combines detection reports, hydrophone locations, sound velocity
 - TSPI: Time Space Position Information



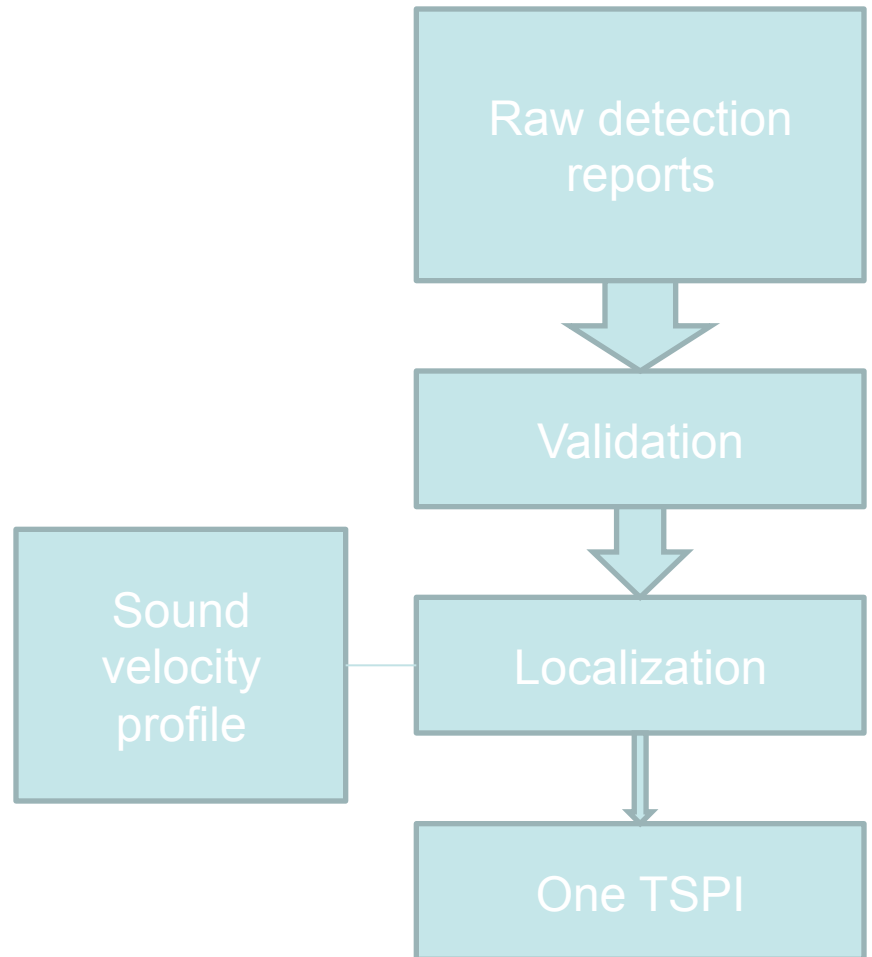
Components: Tracking

- Sound speed
 - Approx. 1500 m/s
- Doppler:
 - 1 knot = 0.51 m/s
- Transmit-receive latency
 - Track delay



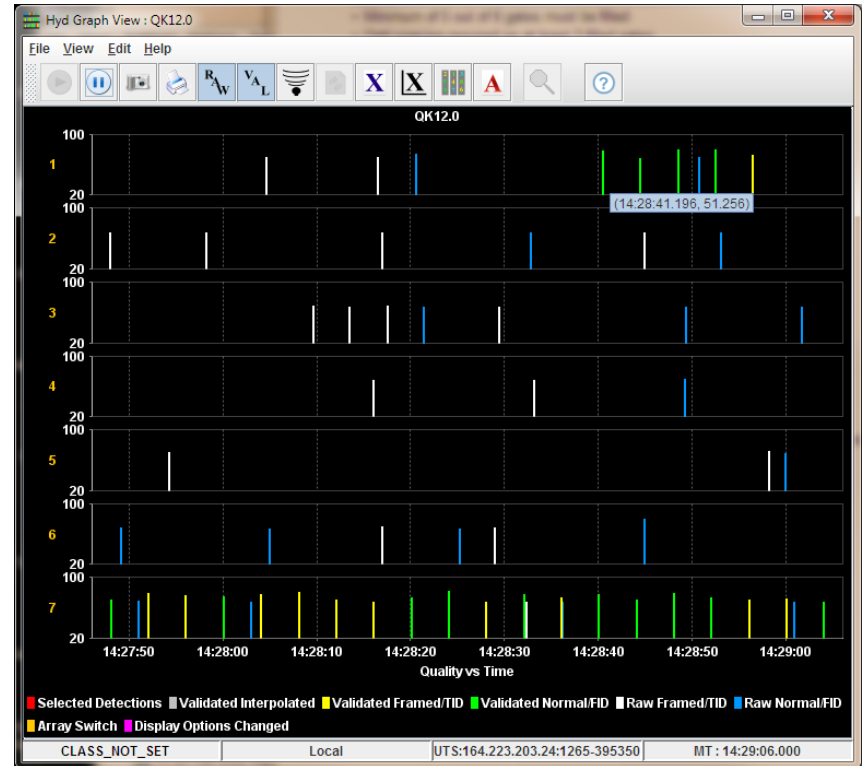
Components: Tracking

- Data flow must be deterministic
 - Two runs with same data => exact same output
- Sound transit delay requires a time window to capture relevant data
- Fundamentally single-threaded



Validation: Removes Extraneous Data

- Per pinger across all phones
- Eliminate possibly bad data
- Remaining is “valid”
- Single-threaded?
 - But each phone is independent...
 - Determinism overrides possible performance



Validation: Sequenced and Framing

- Sequenced Pingers:
 - Up to 12 targets
- Framed
 - A TID and FID pair
 - TID = 1 – 12
 - FID = TID – 12
 - Up to 63 simultaneous targets

DPSK Ping Patterns

Sequenced: (ADL DPSK signal = 19-bit TID + 3-bit SEQ + 4-bit DEPTH)

Ping#	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
SEQ	0	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7
Depth	M	L	L	L	M	L	L	L	M	L	L	L	M	L	L	L

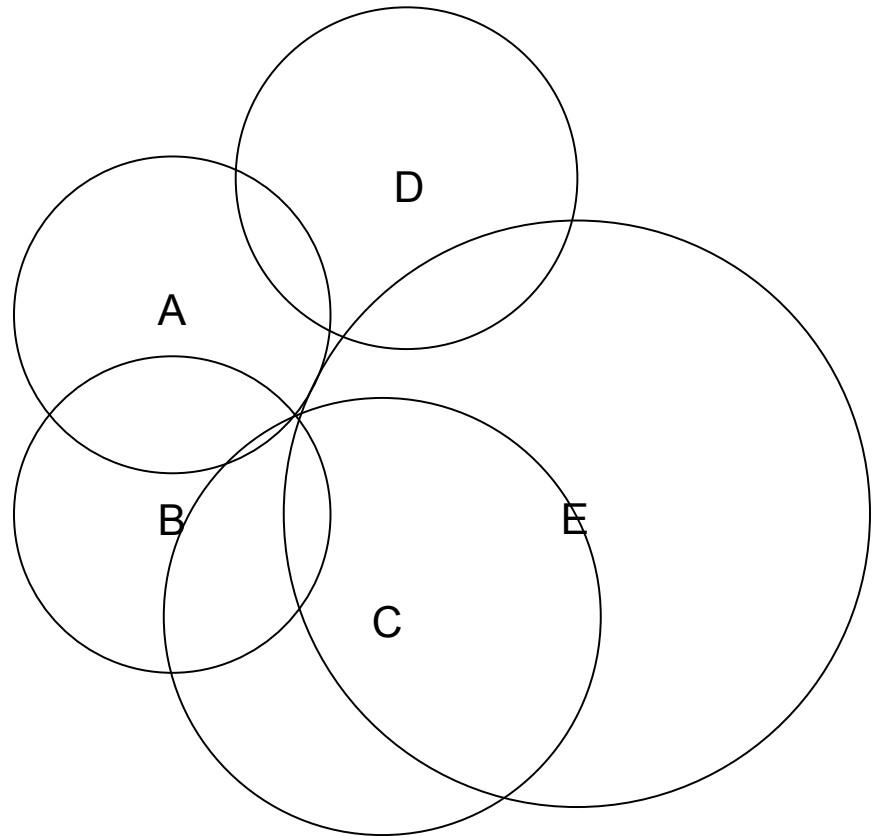
Framed: (SWTR Signal = 7-bit Hamming-encoded DEPTH + <N>bit TID)
(e.g. TID 1, FID 2)

Ping#	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TID	1	1	1	1	2	2	1	2	1	2	2	2	2	1	1	2
Depth	M	L	L	L	M	L	L	L	M	L	L	L	M	L	L	L

Depth Code: 8-bit code sent in (2) 4-bit nibbles
M=Most significant 4-bits, L= Least significant 4-bits

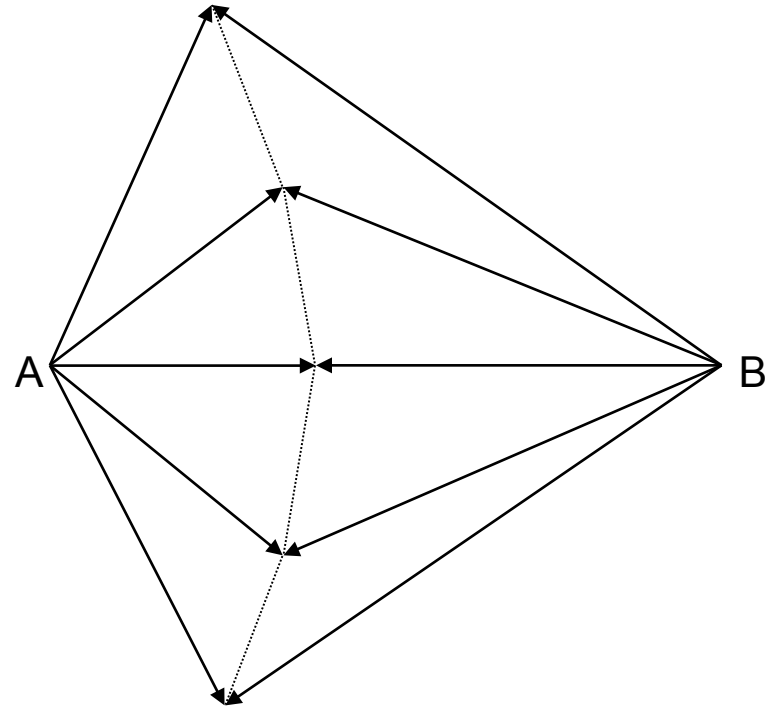
Localization: Where?

- Input: Validated data
- Context:
 - A known set of hydrophone locations
 - Ping ordering
 - Sound velocity profile
- Times of arrival at hydrophones + context = position



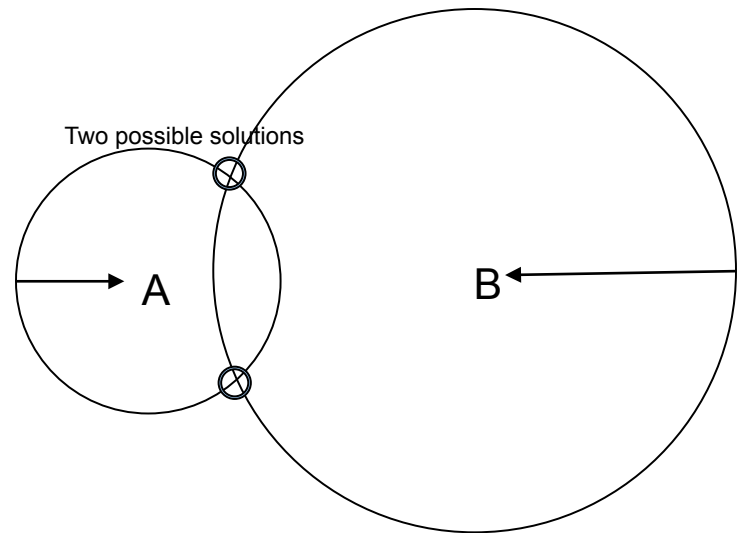
Localization: Hyperbolic

- We do NOT know the time of emission
- TDOA = Time difference of arrival at two phones
- Hyperbola = possible positions that would have identical TDOAs



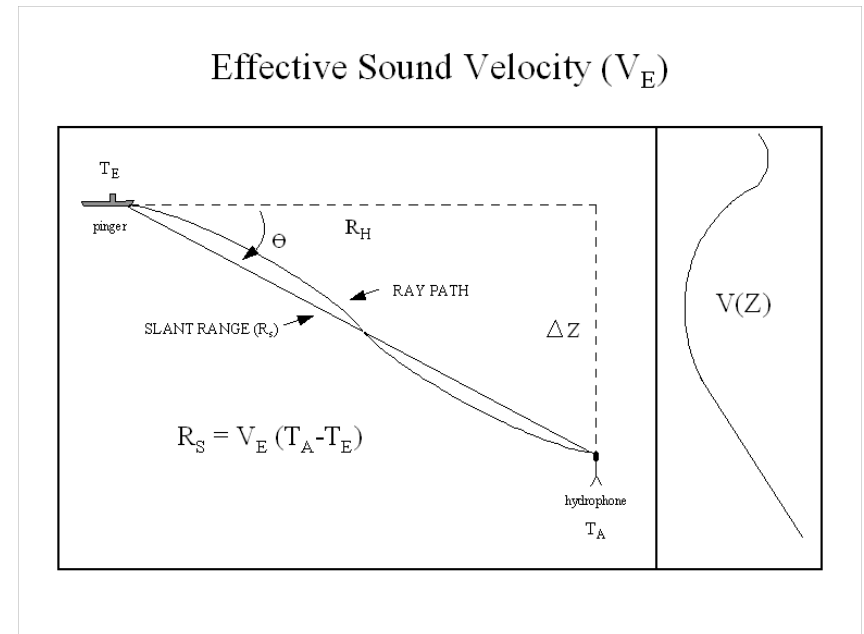
Localization: Spherical

- We DO know the time of emission
 - Given enough time, we can predict
- Spherical radius = $(TOA - TOE) / C$
 - TOA = Time of arrival
 - TOE = Time of emission
 - C = average sound speed over path



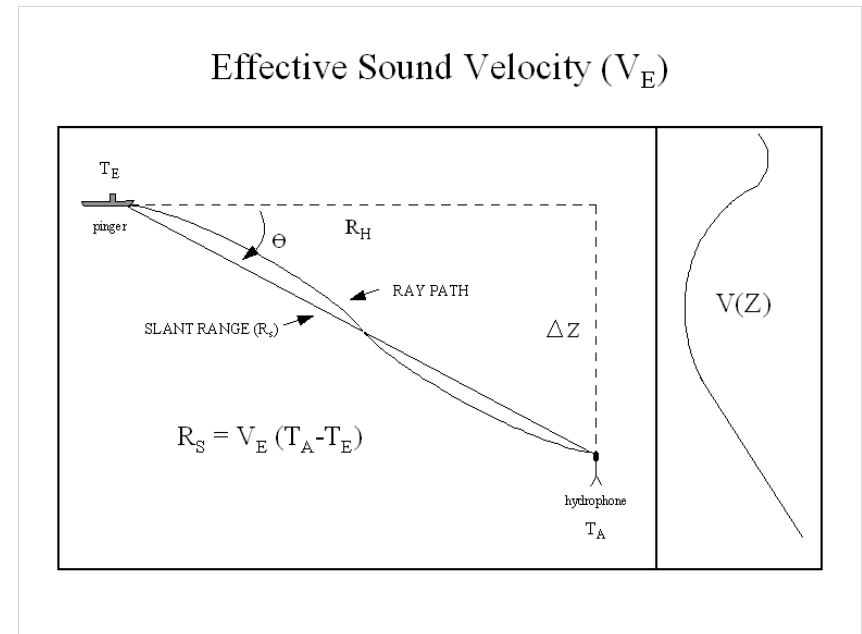
Effective Sound Velocity

- Sound never travels in straight lines
- Path varies with sound speed at depth
- Tracing all the possible ray paths is infeasible



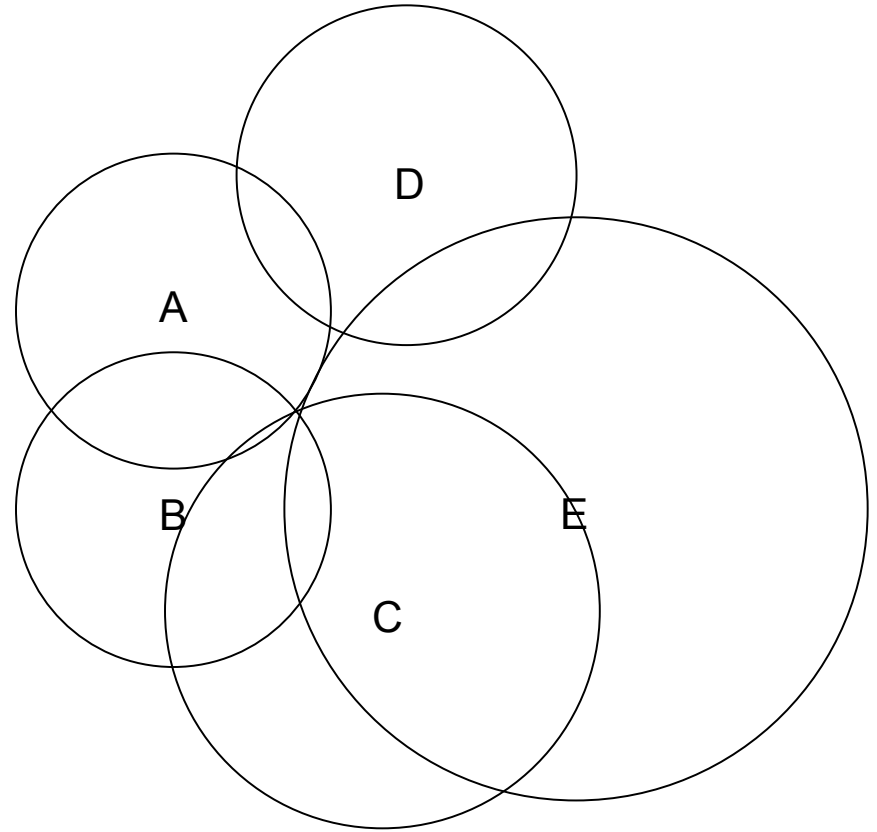
Effective Sound Velocity

- ESV = straight line distance / elapsed transit time
- Pre-computed table captures ray traces
 - A useful engineering approximation
 - Trades space and accuracy for speed
 - Per-month, per-day or per-operation



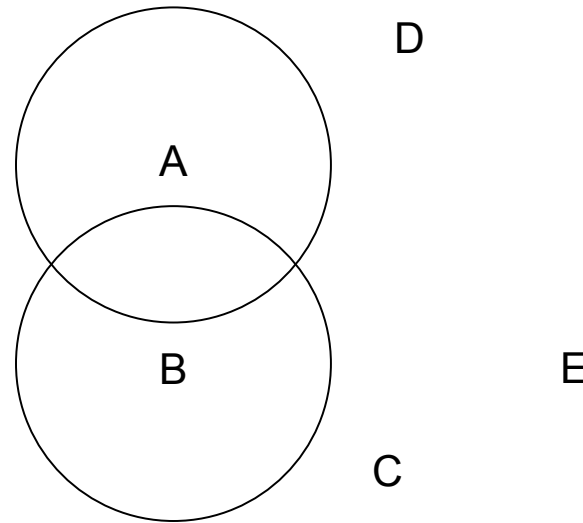
Spherical Tracking

- Time of emission is known
- Geometric options:
 - 3D = 3 degrees of freedom
 - 2D fixed depth = operator specified
 - 2D encoded = on-board depth sensor transmits



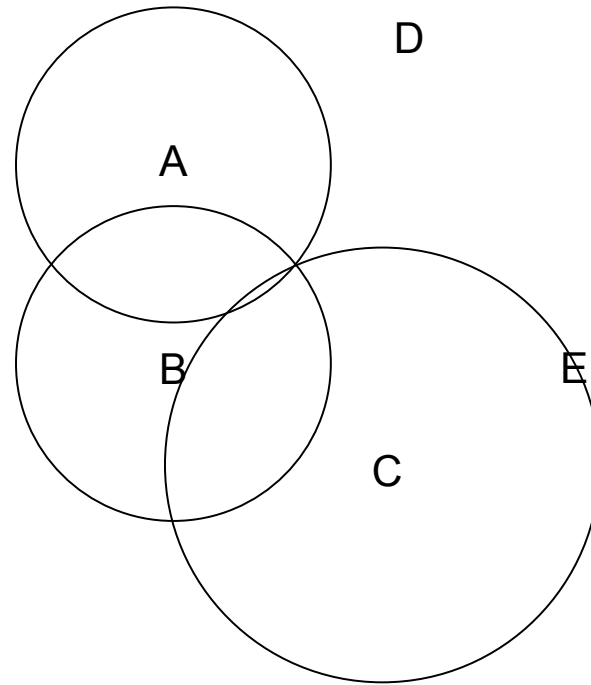
Spherical Tracking

- 2-phones are ambiguous
 - Operator can specify left or right solution
 - Tracking can derive from context



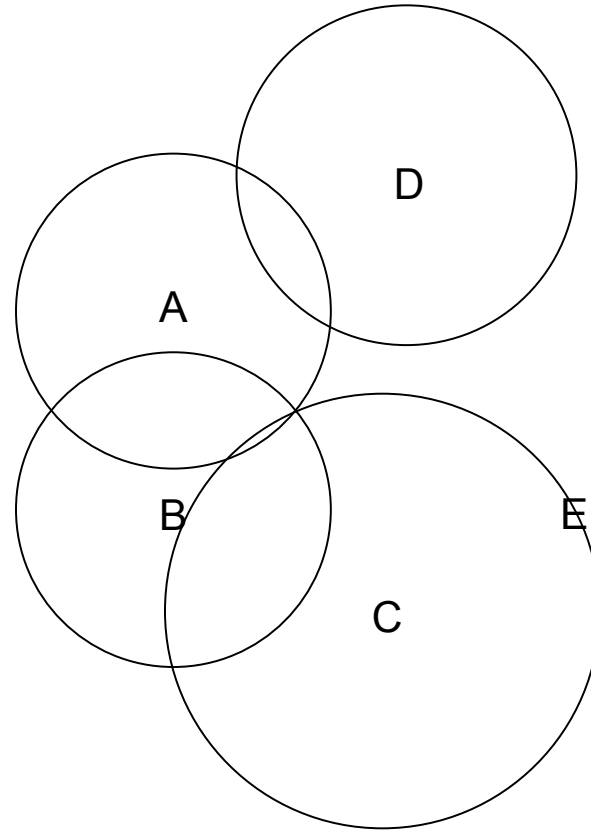
Spherical Tracking

- 3-phones are unambiguous in 2D
 - Sufficient for fixed or encoded depth
 - Still ambiguous in 3D



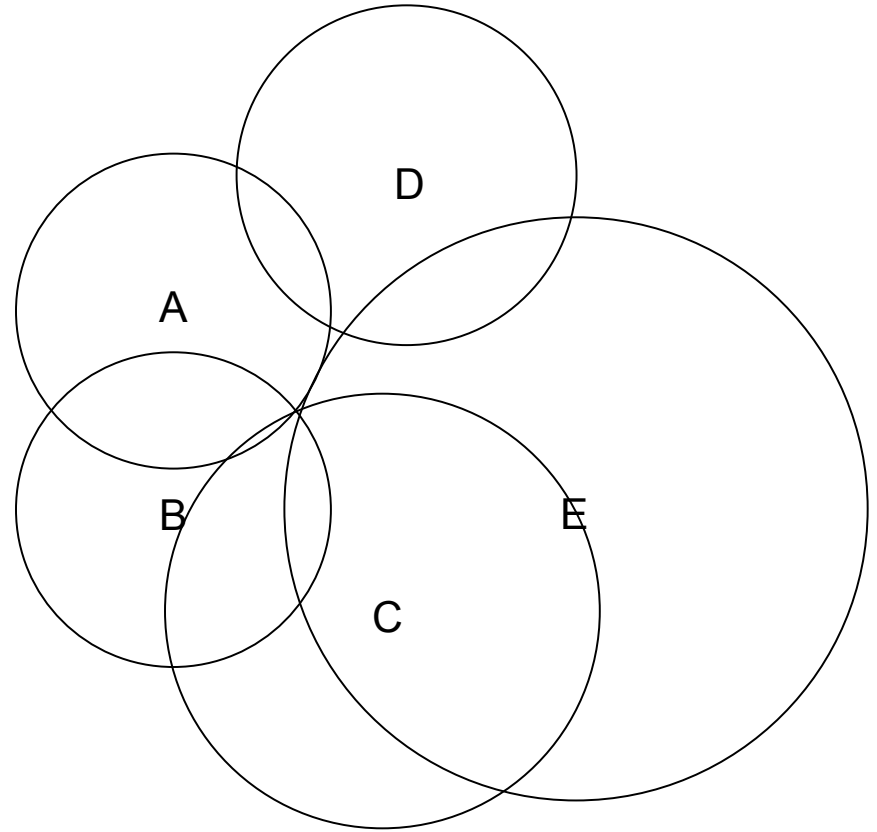
Spherical Tracking

- 4-phones
 - Unambiguous in 3D
 - Error-tolerant in 2D
 - High residual => drop D's detection



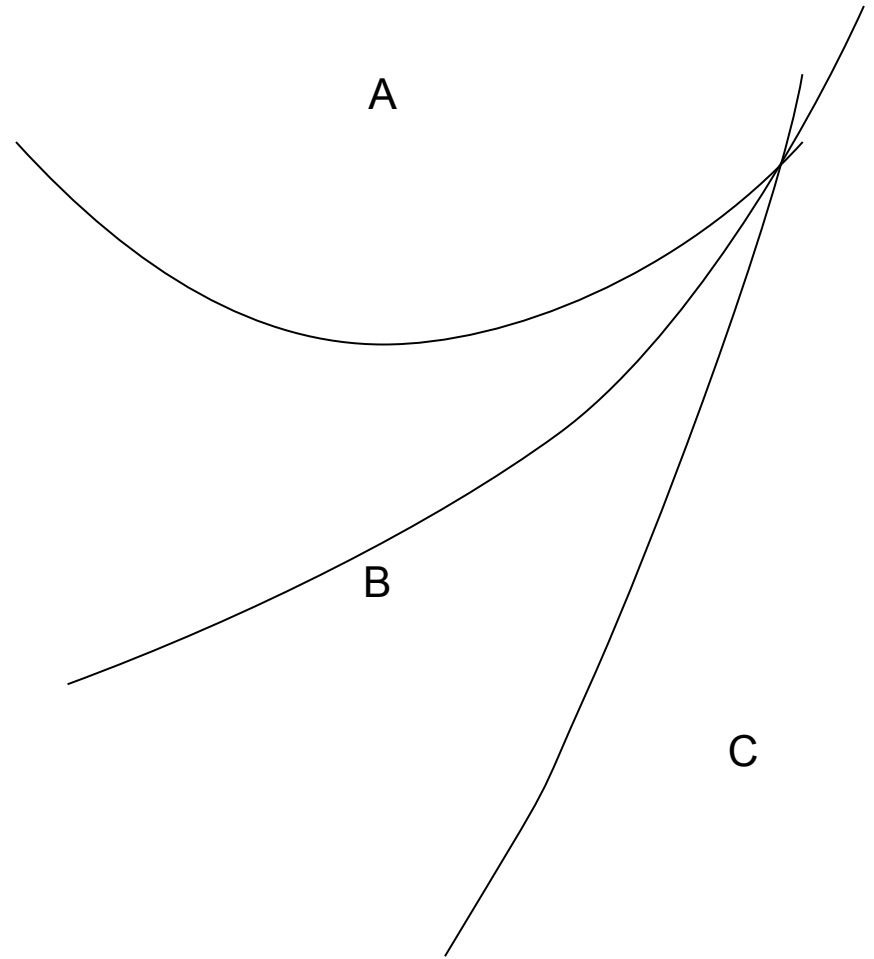
Spherical Tracking

- 5-phones
 - Can optimize geometry for best 2D solution
 - Error tolerant in 3D
 - High residual => drop D's detection



Hyperbolic Tracking

- TOE = unknown
- Curves = paths of equal TDOA
- Requires one more phone than spherical
 - E.g., 3 phones required for 2D track

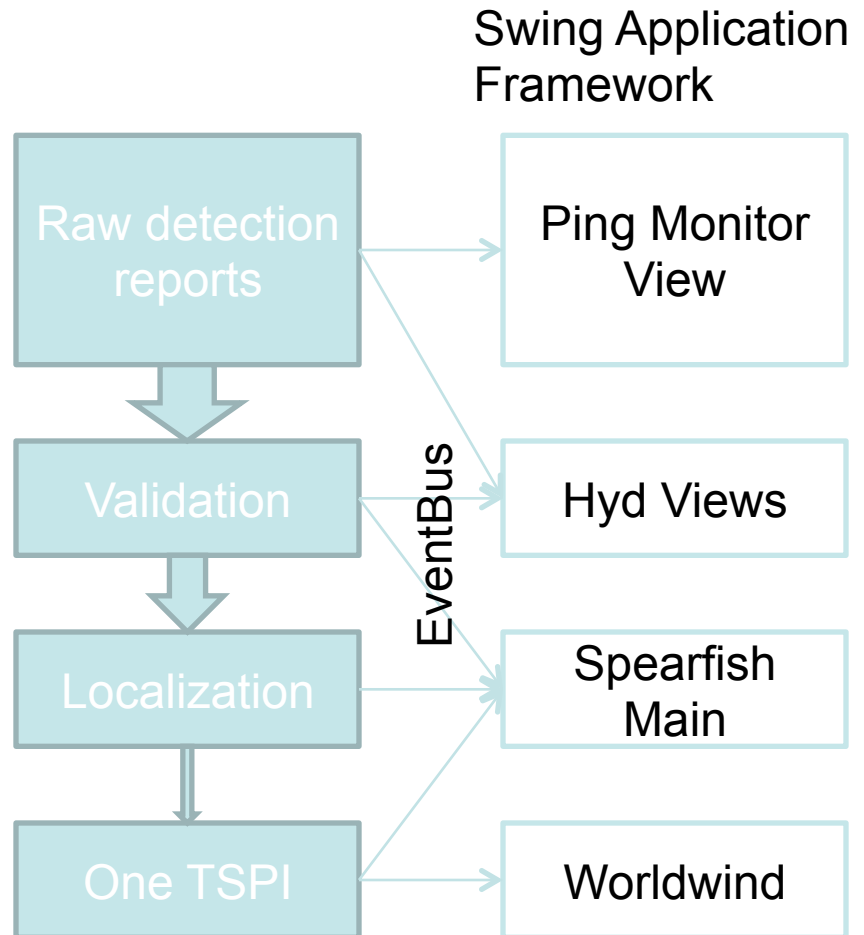


Standard Tracking Scenarios

- Normal running: sub, target, surface ship track easily on range
- Launch: two pingers in close proximity, which one wins?
- End of run: could go vertical, directional bias reduces sound at phone
- On surface: noisy, rolling, perhaps vertical

User Interface

- Intentional redundancy
 - Many slices of same data
 - Many ways to get there from here
 - Many interaction options



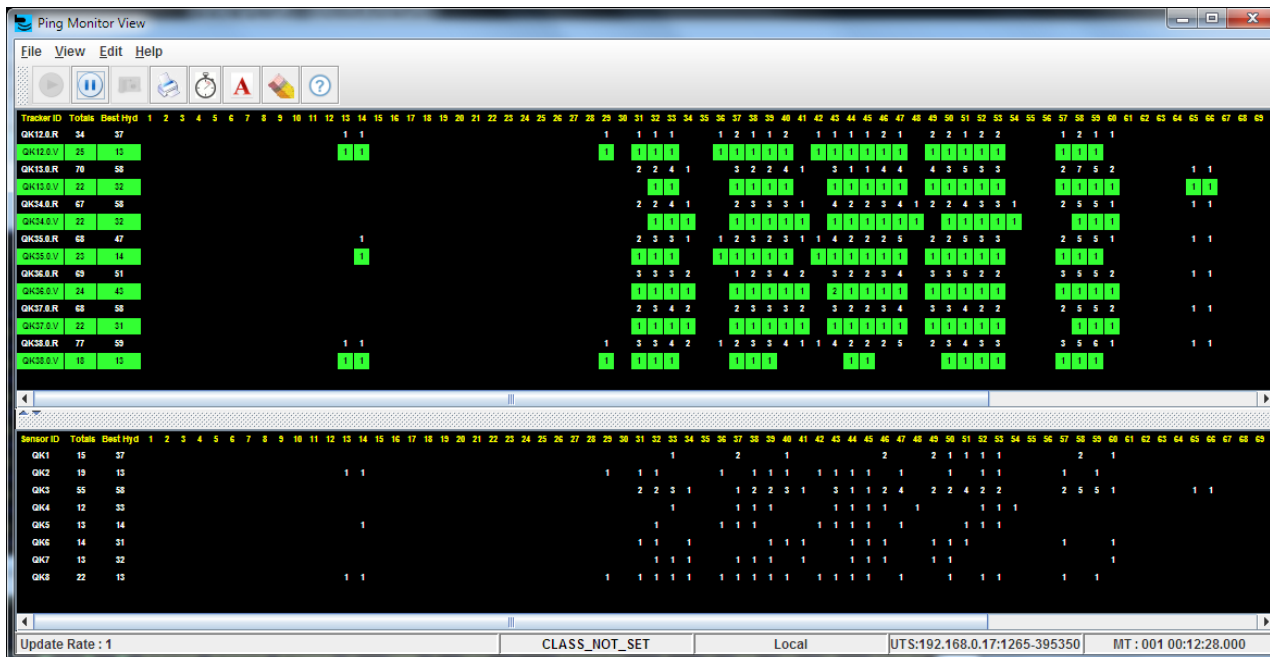
User Interface: Spearfish

- Spearfish Manager = the main window
 - Displays current trackers and controls settings
 - Changes to tracker => tracker control EB topic

Tracker ID	Is Enabled	Is Auto PingPeriod	Status	Emission Time (sec)	Contact Time (sec)	Track Acquis Time (sec)	Ping Period (sec)	Observed Ping Period (sec)	Array ID	Ping Offset (sec)	Hyds In Soln	Hyds In Solution	# of Hyds Validating	GDOP	Crs (deg)	Spd (kts)	Depth (ft)	Latitude (deg)
QK12.0	✓	A	✓	001 00:11:10.992	001 00:00:02.698	001 00:00:00.991	1.0	0.999996	45	-0.00749	7	45 44 38 39 51 46 52	24	3.3	359.99	19.44	-7.99	24 31' 20.569" N
QK34.0	✓	A	✓	001 00:11:11.993	001 00:00:02.748	001 00:00:00.995	1.0	1.000003	46	-0.00618	6	46 45 39 40 52 53	21	2.7	224.98	20.93	225.76	24 30' 31.693" N
QK35.0	✓	A	✓	001 00:11:10.992	001 00:06:41.070	001 00:06:39.993	1.0	0.999975	45	-0.00729	7	45 44 38 51 39 46 52	23	2.9	179.99	1.84	-5.77	24 31' 03.044" N
QK37.0	✓	A	✓	001 00:11:11.993	001 00:10:02.263	001 00:09:59.996	1.0	0.999981	45	-0.00615	7	45 39 46 38 44 51 52	21	2.5	280.55	57.41	298.96	24 31' 01.937" N
QK36.0	✓	A	✓	001 00:11:11.993	001 00:10:01.397	001 00:09:59.992	1.0	1.000003	45	-0.00613	7	45 46 39 51 52 44 38	21	2.4	270	58.39	299.58	24 30' 49.303" N
QK38.0	✓	A	✓	001 00:11:11.993	001 00:10:01.397	001 00:09:59.993	1.0	1.000002	45	-0.00616	7	45 39 46 38 44 51 52	22	2.8	289.98	56.95	298.31	24 31' 12.698" N
QK13.0	✓	A	✓	001 00:11:11.993	001 00:10:01.548	001 00:09:59.992	1.0	1.000001	45	-0.00627	7	45 51 44 52 46 38 39	22	3.1	180.01	38.92	236.98	24 30' 10.709" N

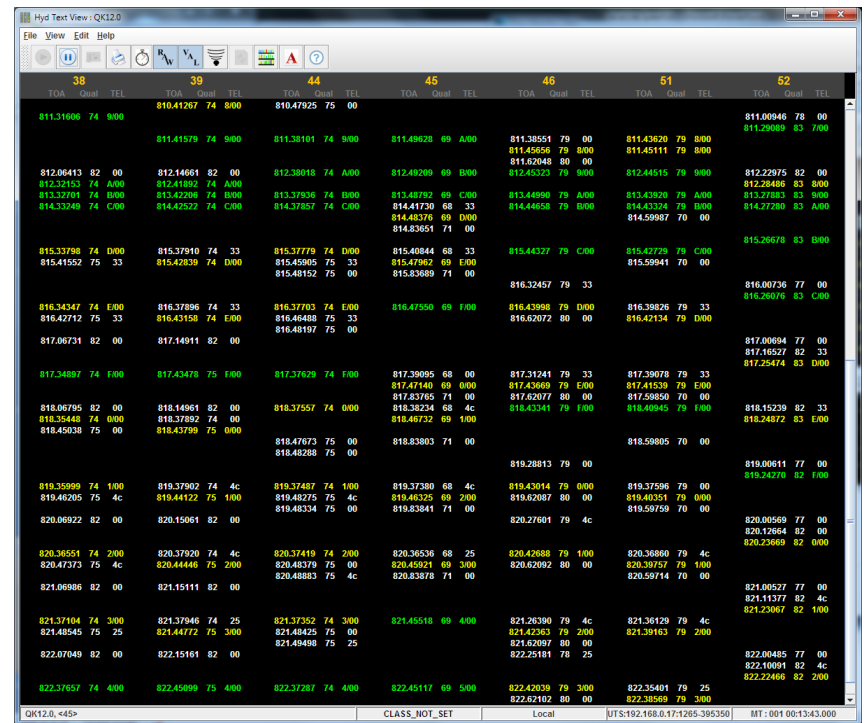
Data Manager Received CLASS_NOT_SET Local UTS:192.168.0.17:1265-395350 MT: 001 00:11:18.000

- Raw detection report EB topic
- Is tracking receiving data?
- Is tracking receiving valid data?



User Interface: HydTextView

- Raw and validated detection report topics
 - Filtered by tracker
 - Time of arrival
 - Quality
 - Telemetry
 - Validity = color
 - Hydrophone

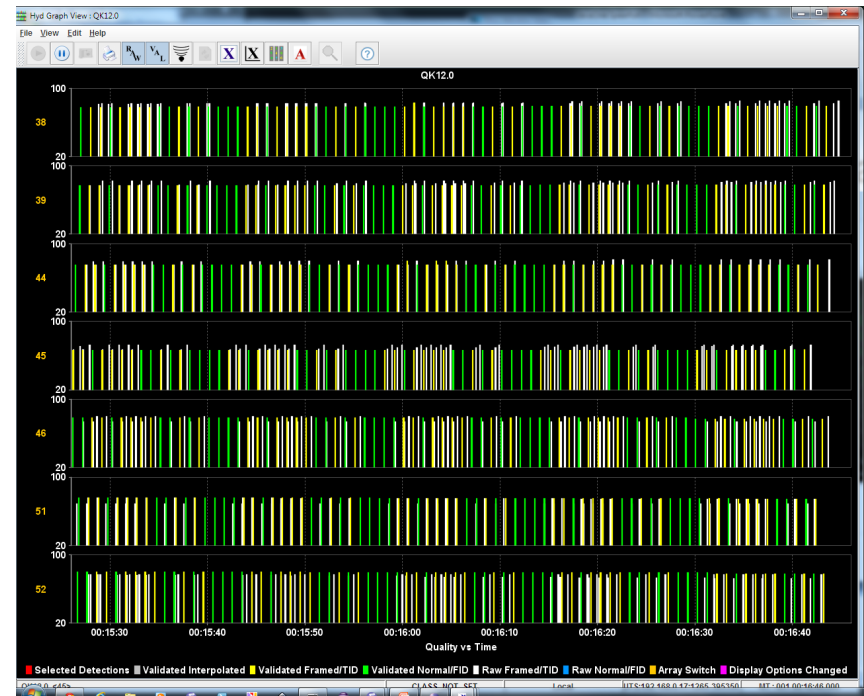


The screenshot shows the 'Hyd Text View' application window. The title bar reads 'Hyd Text View v0.12.0'. The menu bar includes 'File', 'View', 'Edit', and 'Help'. The toolbar contains icons for file operations, search, and display settings. The main content area displays a table of detection data organized into columns for different trackers (38, 39, 44, 45, 46, 51, 52). Each column has sub-headers for TOA, Qual, and TEL. The data rows contain numerical values, some of which are color-coded (green, red, blue) to represent validity. The status bar at the bottom shows 'OK12.0, <45>', 'CLASS_NOT_SET', 'Local', 'JTS:192.168.0.17:1265-395350', and 'MT: 001 00:1343.000'.

38			39			44			45			46			51			52		
TOA	Qual	TEL	TOA	Qual	TEL	TOA	Qual	TEL	TOA	Qual	TEL	TOA	Qual	TEL	TOA	Qual	TEL	TOA	Qual	TEL
811.31606	74	900	810.41267	74	800	810.47925	75	00										811.08946	78	00
			811.41579	74	900	811.38101	74	900	811.49628	69	A00	811.38551	79	00	811.43620	79	800	811.29089	83	700
									811.45656	79	800	811.45656	79	800	811.45111	79	800			
									811.62045	80	00	812.45323	79	900	812.44515	79	900	812.22975	82	00
812.06413	82	00	812.14661	82	00	812.38018	74	A00	812.49209	69	B00	813.48792	69	C00	813.48980	79	A00	812.28486	83	800
812.32153	74	A00	812.41882	74	A00	812.37936	74	B00	814.41730	68	33	814.44658	79	B00	814.43324	79	B00	812.32183	83	800
813.32191	74	800	814.42206	74	800	814.37857	74	C00	814.48376	69	D00	814.59987	70	00				814.33249	74	C00
814.33249	74	C00	814.42527	74	C00				814.83651	71	00							815.26678	83	B00
815.33798	74	D00	815.37910	74	33	815.37779	74	D00	815.40844	68	33	815.44327	79	C00	815.42729	79	C00			
815.41552	75	33	815.42639	74	D00	815.45905	75	33	815.45962	69	E00	815.53689	71	00	815.59941	70	00			
						815.48152	75	00				816.32457	79	33				816.00736	77	00
												816.62072	80	00	816.39826	79	33	816.26076	83	C00
816.34347	74	E00	816.37896	74	33	816.37702	74	E00	816.47550	69	F00	816.43988	79	D00	816.42134	79	D00			
816.42712	75	33	816.43158	74	E00	816.46488	75	33												
817.00731	82	00	817.14911	82	00	816.48197	75	00										817.08094	77	00
																		817.16527	82	33
																		817.25474	83	D00
817.34887	74	F00	817.43478	75	F00	817.37629	74	F00	817.39095	68	00	817.31241	79	33	817.39078	79	33			
									817.47140	69	000	817.43668	79	E00	817.41539	79	E00			
									817.83765	71	00	817.62077	80	00	817.59850	70	00			
818.06785	82	00	818.14961	82	00	818.37557	74	000	818.38234	68	4c	818.43341	79	F00	818.40845	79	F00	818.15239	82	33
818.35448	74	800	818.37892	74	00				818.40732	69	100							818.24872	83	E00
818.45038	75	00	818.43799	75	000															
						818.47673	75	00	818.83803	71	00				818.59805	70	00			
						818.48208	75	00				819.28813	79	00				819.00611	77	00
																		819.24270	82	F00
819.35999	74	100	819.37902	74	4c	819.37487	74	100	819.37380	68	4c	819.43014	79	000	819.37596	79	00			
819.45205	75	4c	819.44122	75	100	819.48275	75	4c	819.48325	69	200	819.52087	80	00	819.40351	79	000			
						819.48334	75	00	819.83841	71	00	820.27601	79	4c	819.59759	70	00			
820.06922	82	00	820.15061	82	00													820.00569	77	00
																		820.32664	82	00
820.36551	74	200	820.37920	74	4c	820.37419	74	200	820.36536	68	25	820.42688	79	100	820.36860	79	4c	820.23669	82	000
820.47373	75	4c	820.44446	75	200	820.48379	75	00	820.45921	69	300	820.62092	80	00	820.59714	70	00			
						820.48893	75	4c	820.53878	71	00									
821.06986	82	00	821.15111	82	00													821.00527	77	00
																		821.11377	82	4c
821.37104	74	300	821.37946	74	25	821.37352	74	300	821.45518	69	400	821.26390	79	4c	821.38129	79	4c	821.25067	82	100
821.48545	75	25	821.44772	75	300	821.48425	75	00				821.42363	79	200	821.39163	79	200			
822.07049	82	00	822.15161	82	00	821.49498	75	25				821.62087	80	00						
												822.25181	78	25				822.00485	77	00
																		822.10091	82	4c
822.37657	74	400	822.45999	75	400	822.37287	74	400	822.45117	69	500	822.42039	79	300	822.35401	79	25	822.24466	82	200
												822.62102	80	00	822.38569	79	300			

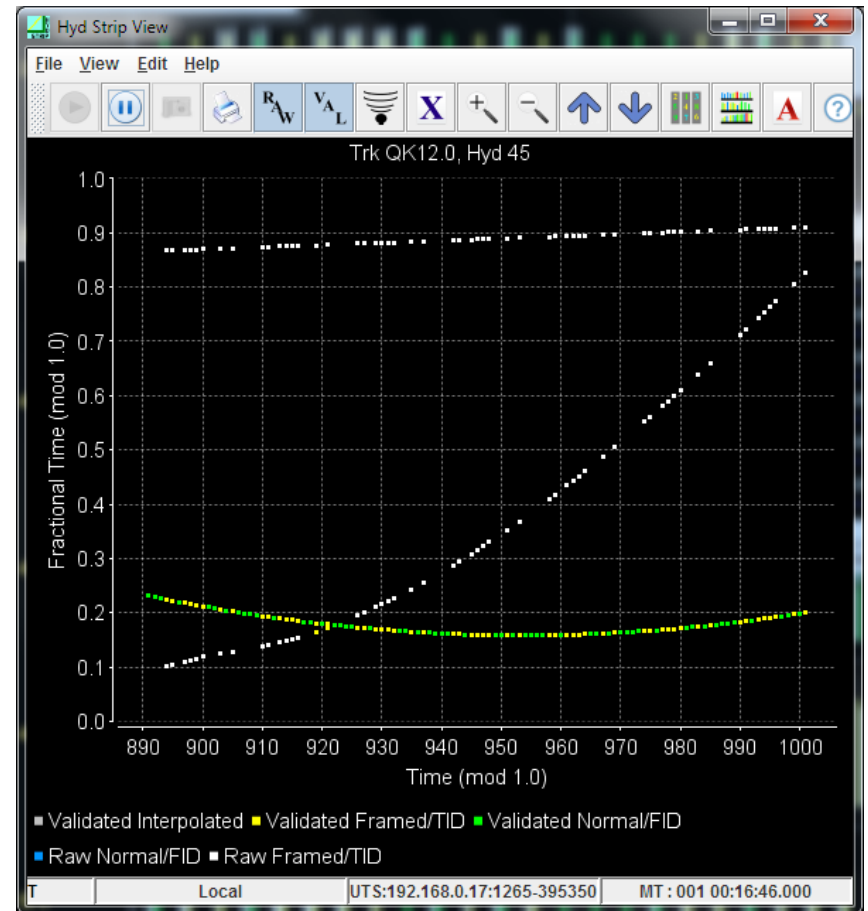
User Interface: HydGraphView

- Raw and validated detection report topics
 - Time series of same data
 - Quality = height
 - Oldest on left, newest on right



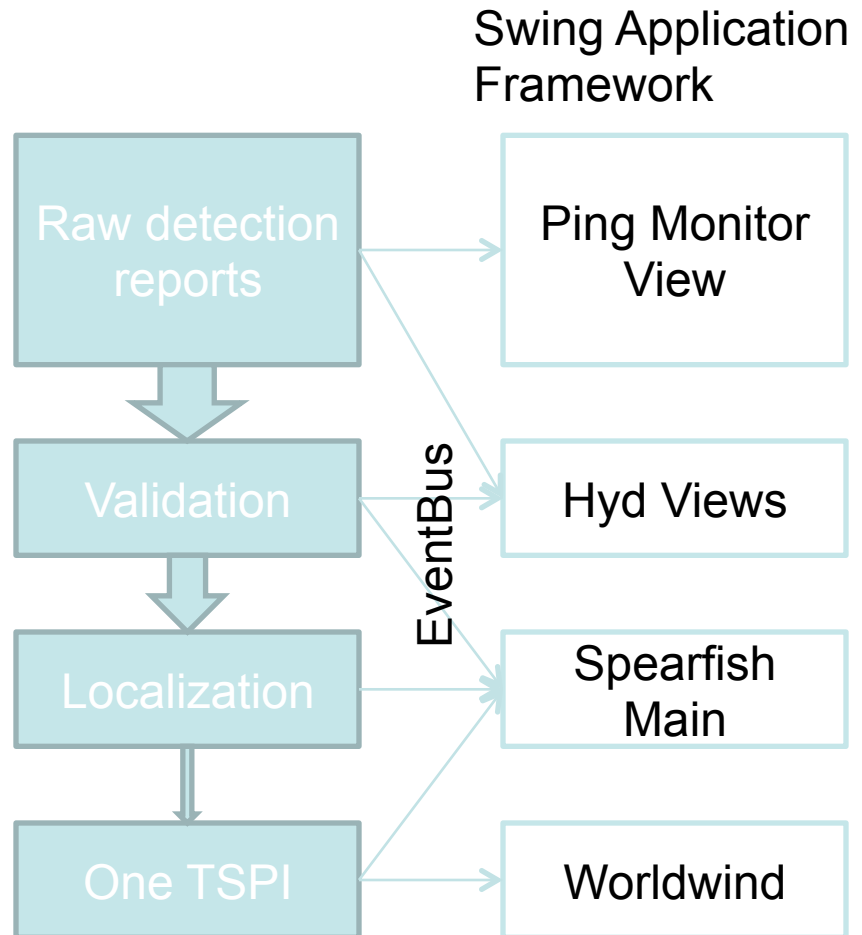
User Interface: HydStripView

- Same data as other HydViews
 - Single hydrophone
 - X-axis: Arrival time
 - Y-axis: Fraction of rep rate
 - Later pings appear higher on chart



Concurrency: 1.0 to 7

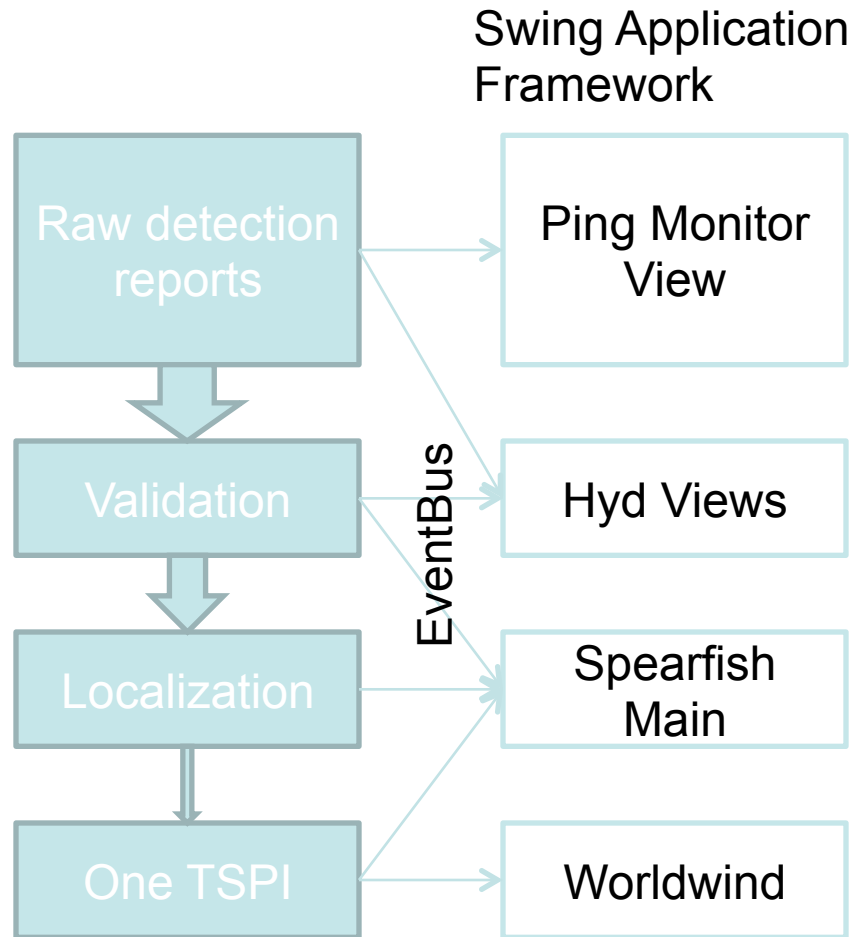
- Java 1.0 to 7
 - Initial work began in mid 90s
 - Under continuous development since
- Correctness reminder:
 - No data loss
 - GUI & processing should not interfere



Concurrency: RMI vs EventBus

- RMI = GUI and processing impact

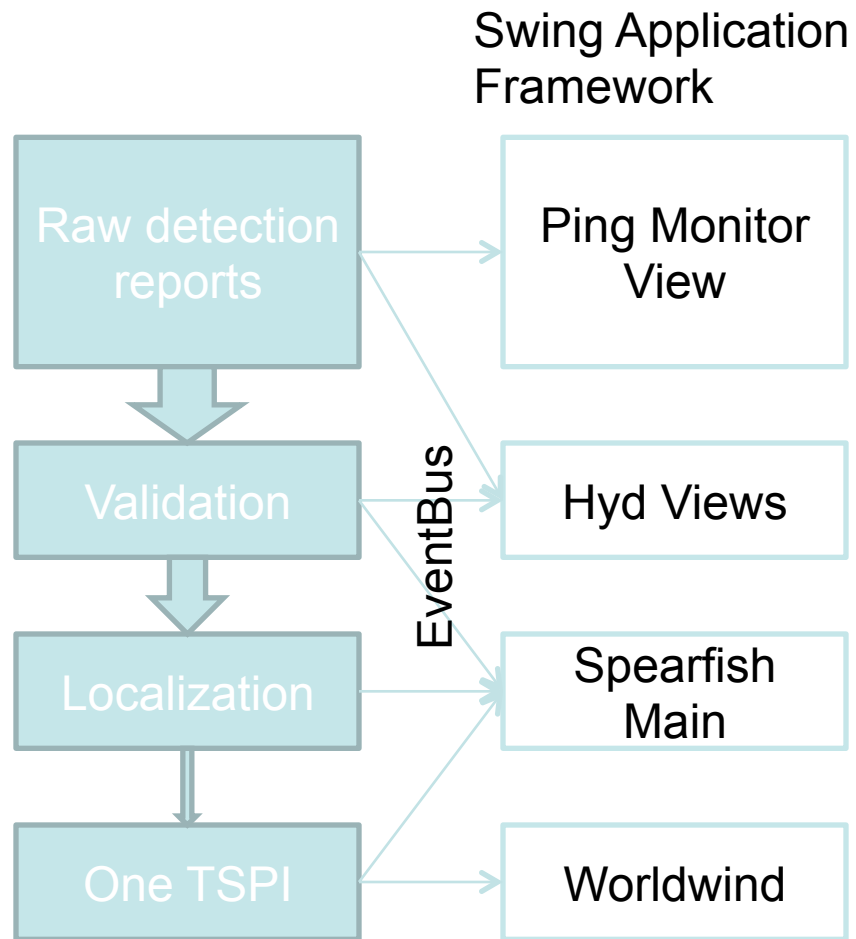
```
display() {  
    // RMI call blocks GUI  
    validation.getDR();  
    // Processing and  
    // Swing threads  
    // coupled  
    chart.showDR();  
}
```



Concurrency: RMI vs EventBus

- EventBus = send data fast, display as / when possible
 - Some contention but locking slows processing

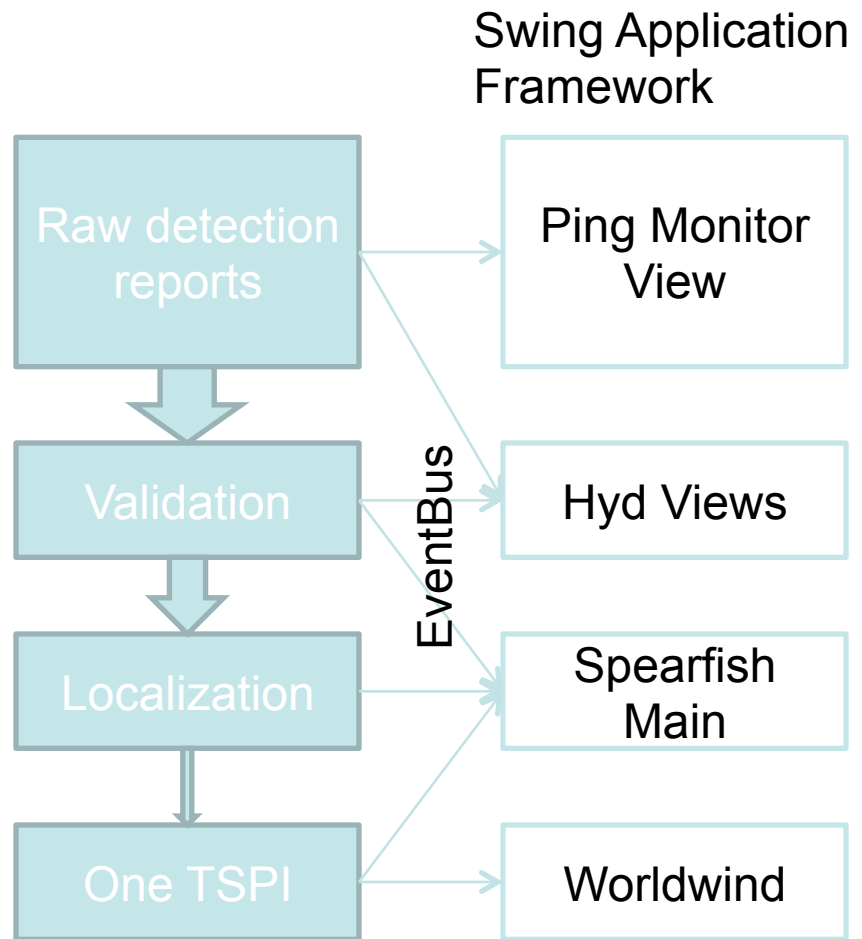
```
onEvent(dr) { // EB thread
    // Contention
    list.add(dr);
}
display() { // Swing thread
    // Contention
    timeSeries.add(list);
    chart.display(timeSeries);
}
```



Concurrency: CopyOnWrite

- CopyOnWrite
 - No corruption
 - No locking

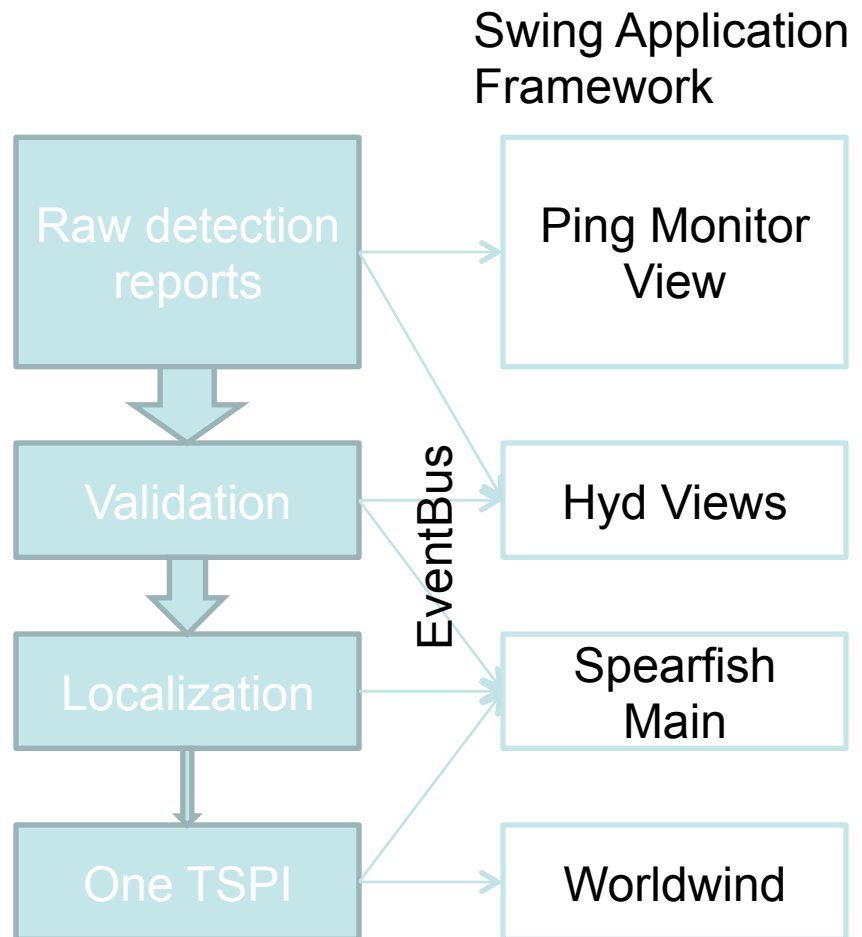
```
onEvent(dr) { // EB thread
    // No contention
    cowList.add(dr);
}
display() { // Swing thread
    // Diff copy from above
    timeSeries.add(cowList);
    chart.display(timeSeries);
}
```



Concurrency: invokeLater()

- `SwingUtilities.invokeLater()`
 - Decoupling of data & display
 - Display when feasible
 - Processing to runs ahead

```
onEvent(dr) { // EB thread
    cowList.add(dr);
    SwingUtilities.invokeLater(
        new Runnable() {
            run() { // Swing thread
                timeSeries.add(cowList);
                chart.display(timeSeries);
            };
        });
}
```



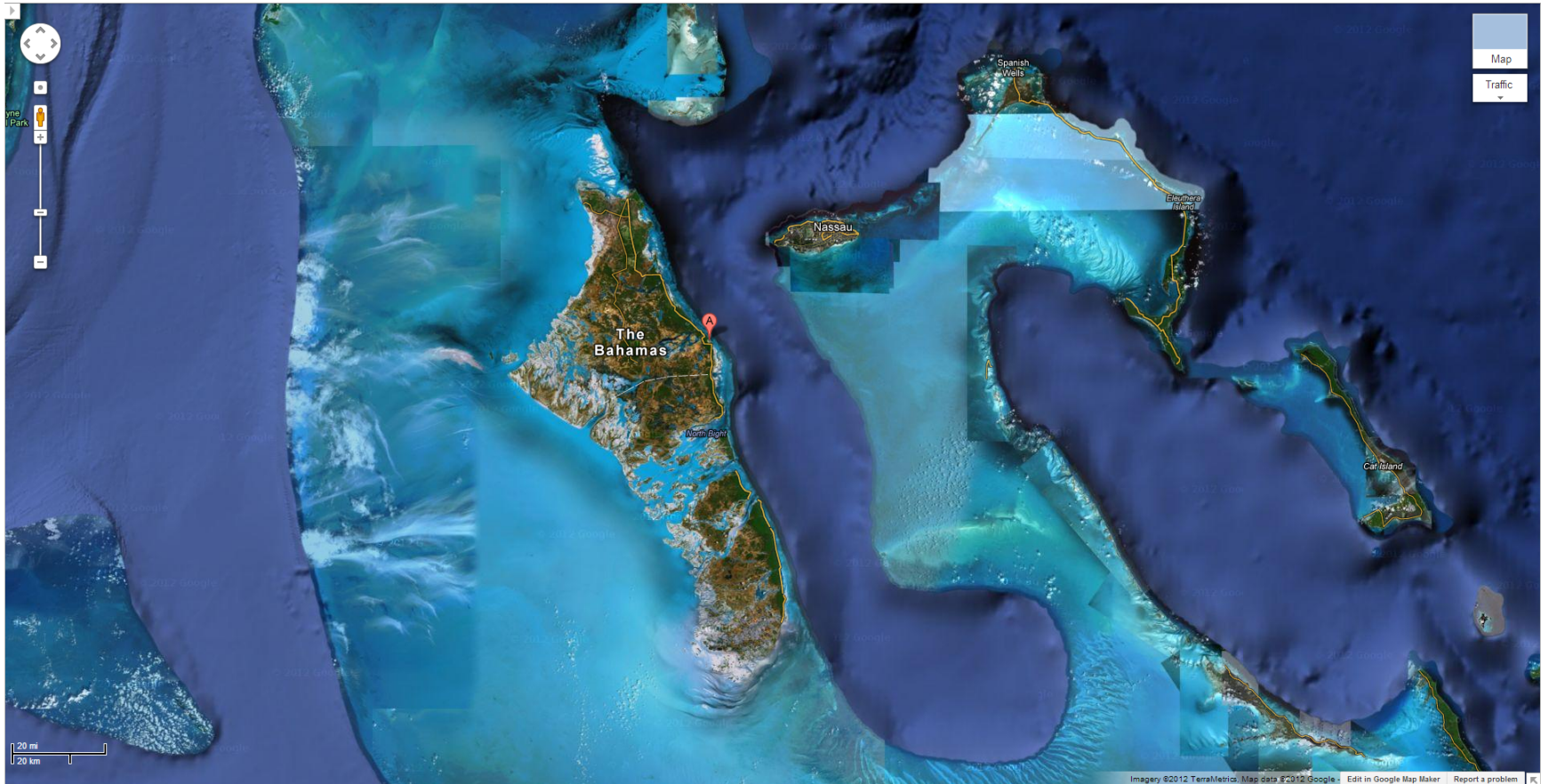
Concurrency: Basics?

- Why discuss relatively basic concurrency mechanisms?
- Large Java software systems' concurrency correctness tends to be inversely proportional to age
 - 1997 = Doug Lea's first edition
 - Large software = large refactoring cost
- Java 7: remediation without heavy refactoring or third party resources
 - E.g., `CopyOnWrite` + `invokeLater()` => more correct without large changes to structure

Live Demonstration

- UNCLASSIFIED
- Data is fictional
- Surface vs. Sub Exercise
- Six weapons launched
- 2 knot surface current
- Total time = 16:40 (run at > 5x speed)

Demo



Bob Cross

Robert.a.cross1@navy.mil

Naval Undersea Warfare Center
Public Affairs Office
1176 Howell Street
Newport, RI, 02841
(401) 832-7742