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

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

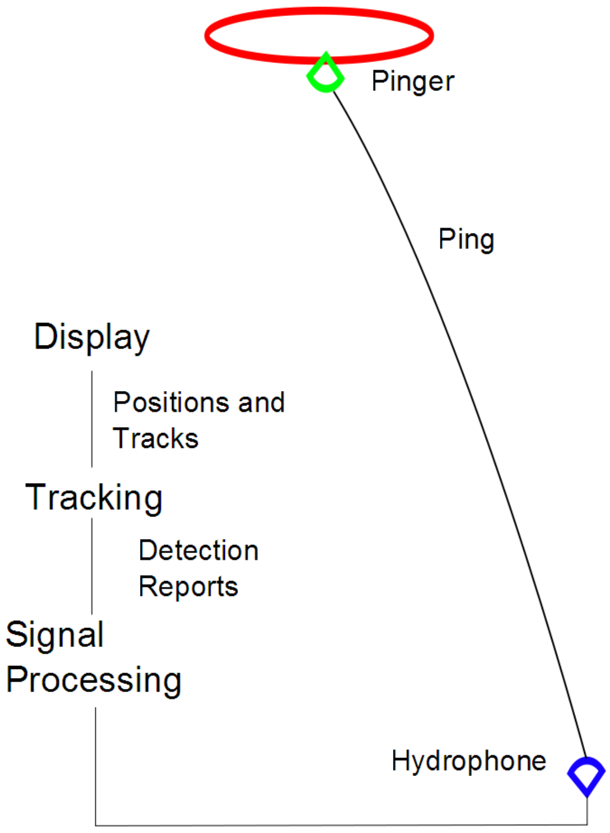
- Training (e.g., “war games”): post-exercise tactical analysis and range safety
- Test and evaluation: absolute and relative accuracy critical
- Novel systems: reconfigure existing capabilities to support new requirements

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

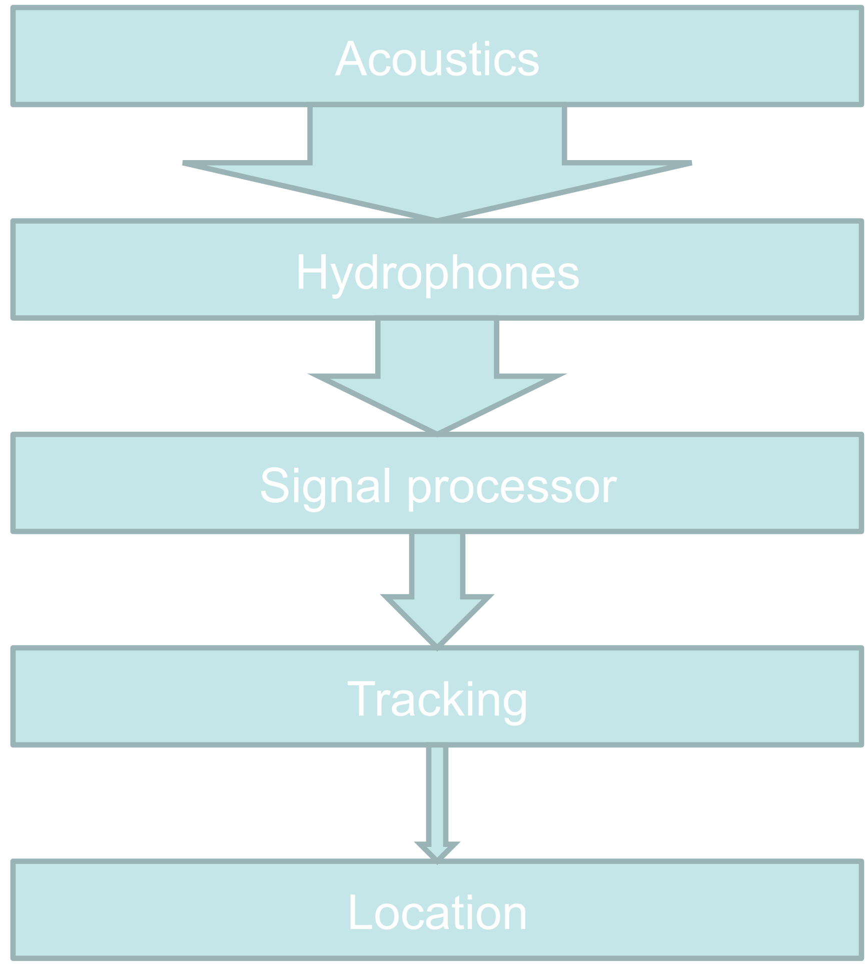
- Range safety
- Detect and track range participants
- Graceful scaling and degradation
- Alert on error conditions
- Accuracy: relative and absolute
- Real-time

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

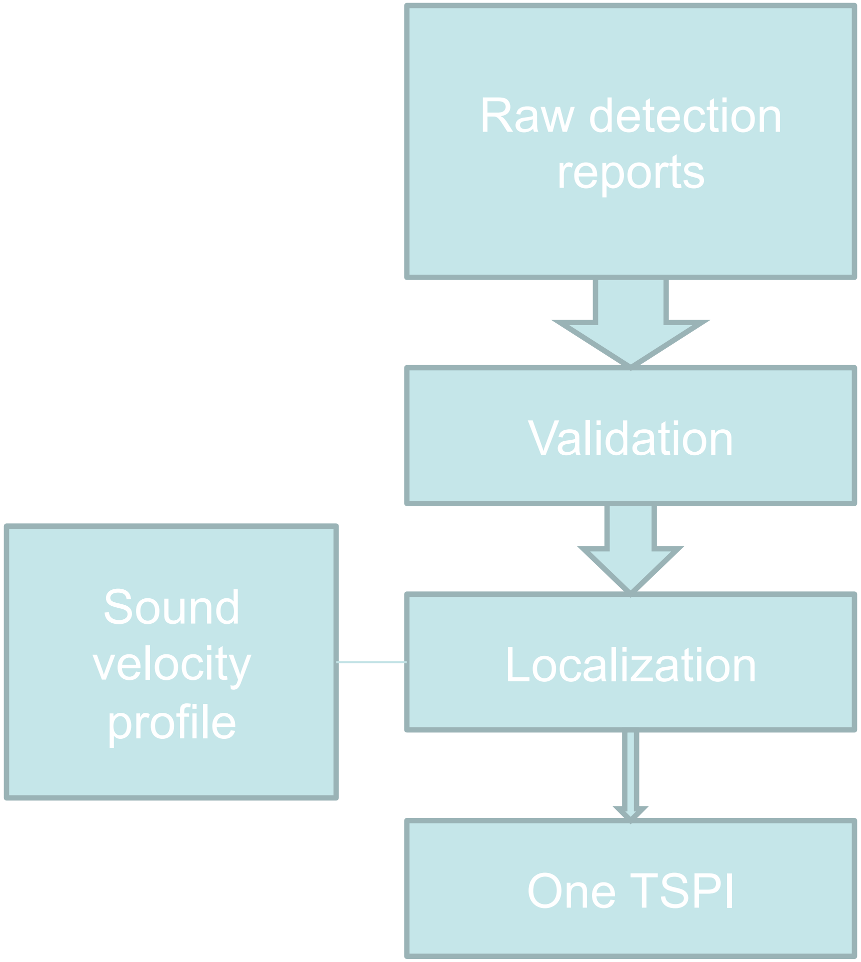
- A splash is anything that is not a ping
- Examples:
  - Broadband impacts
  - Mechanical transients
  - Active emissions
  - Mammals

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

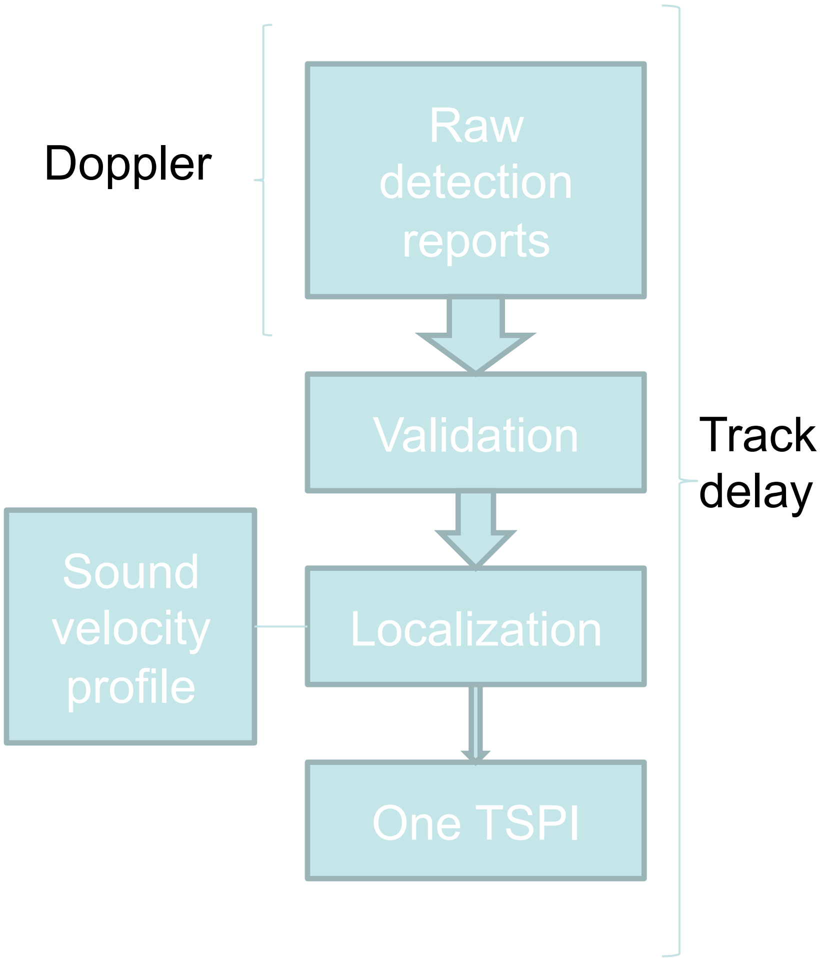
Components: Tracking

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



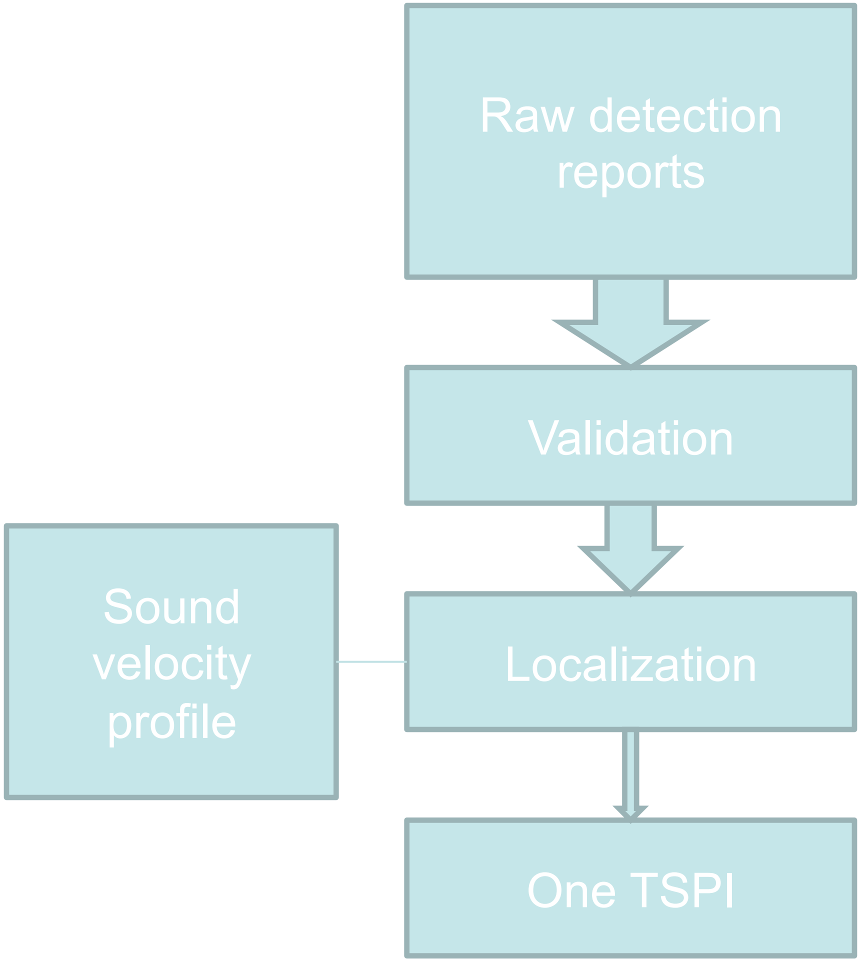
Components: Tracking

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



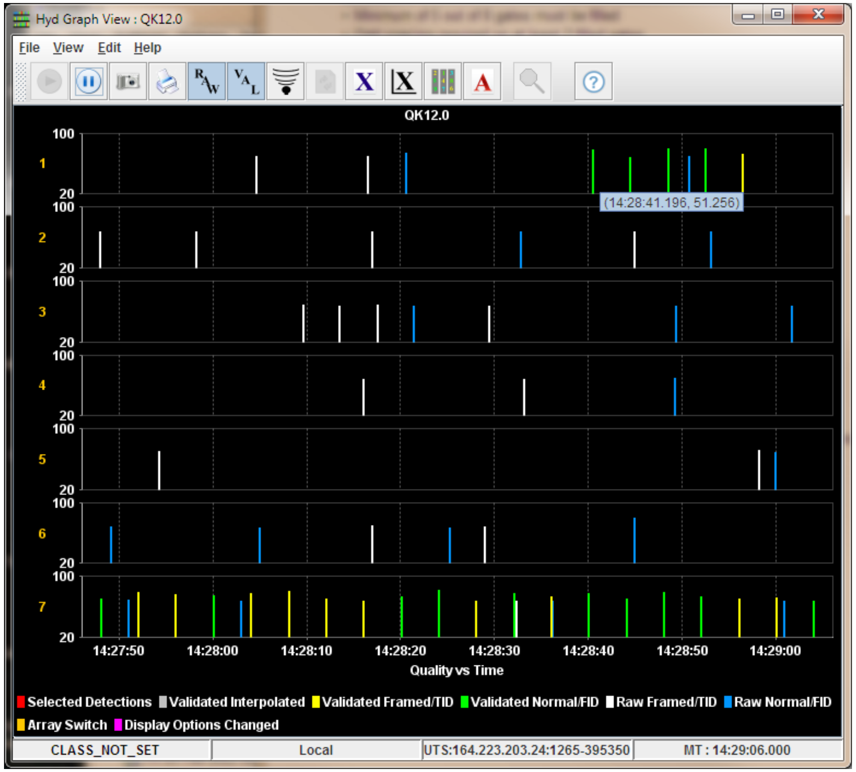
Components: Tracking

- 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

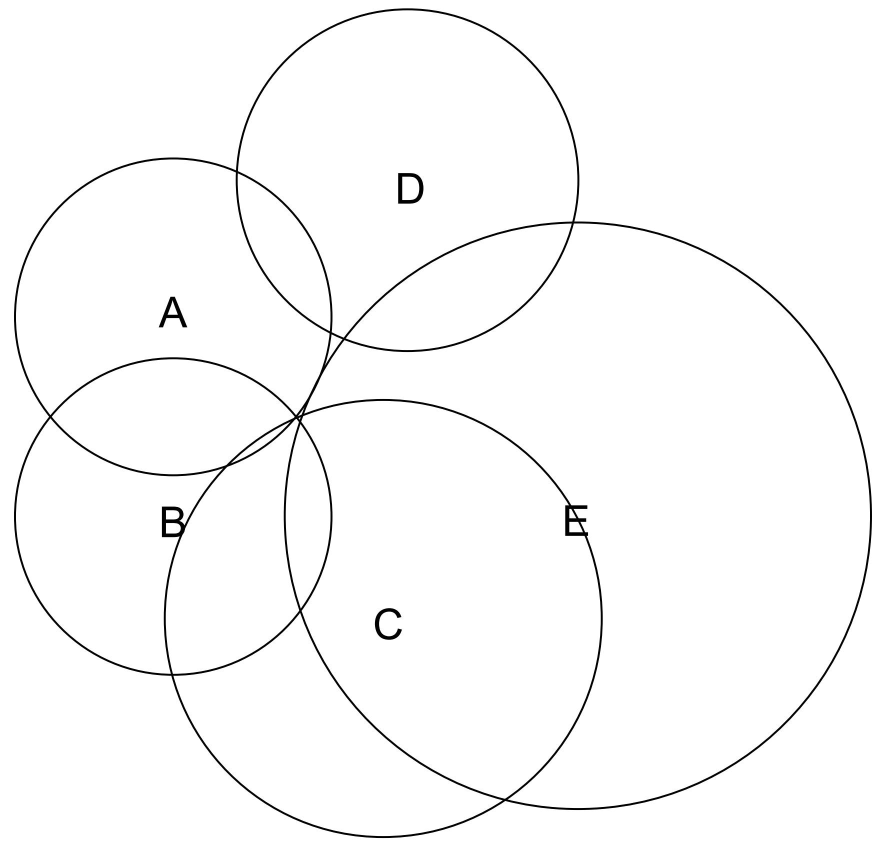
Framed: (SWTR Signal = 7-bit Hamming-encoded DEPTH + <N>bit TID)
(e.g. TID 1, FID 2)

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

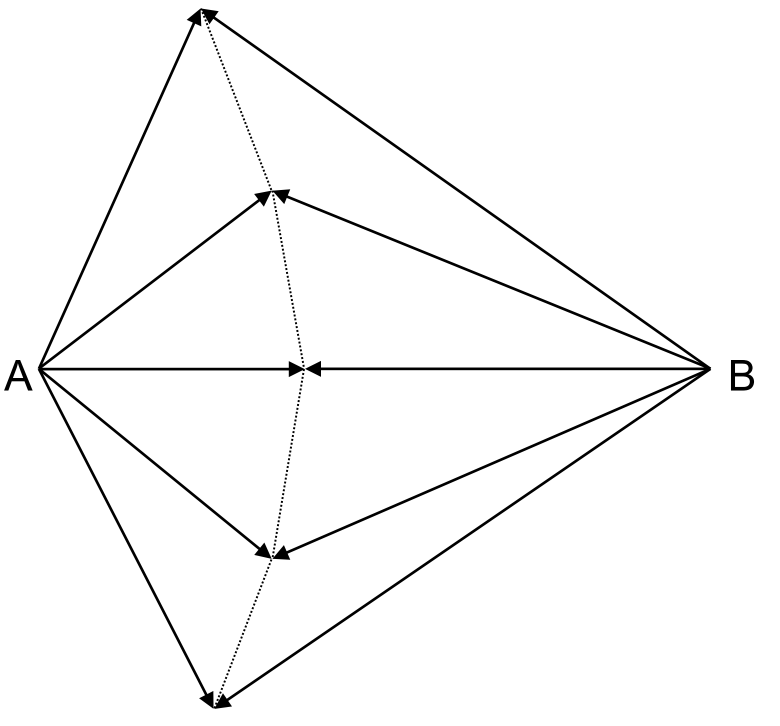
Localization: Where?

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



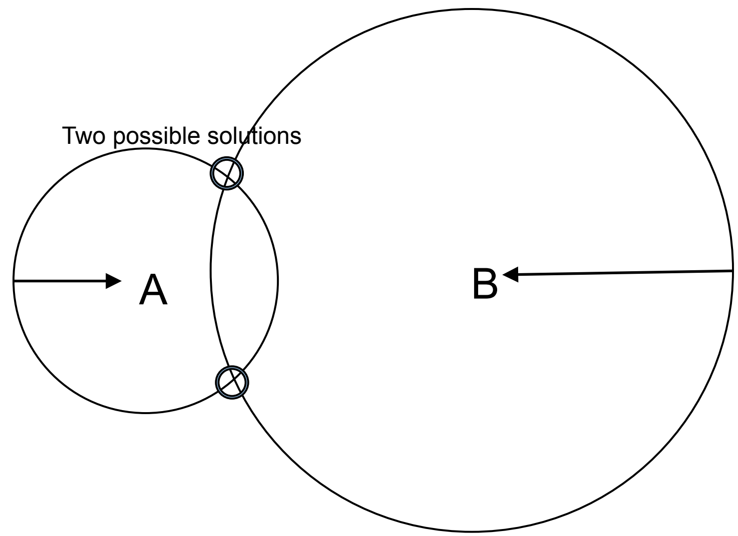
Localization: Hyperbolic

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



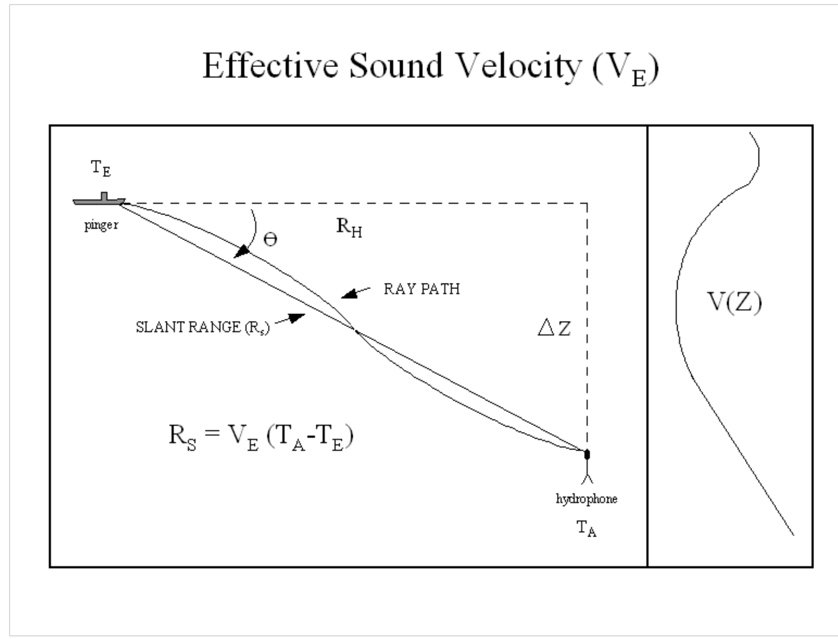
Localization: Spherical

- 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



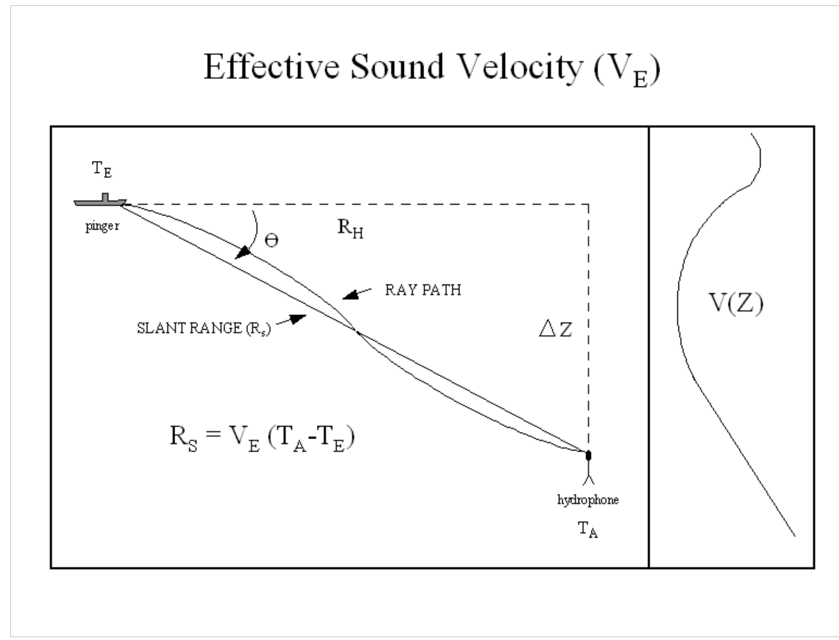
Effective Sound Velocity

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



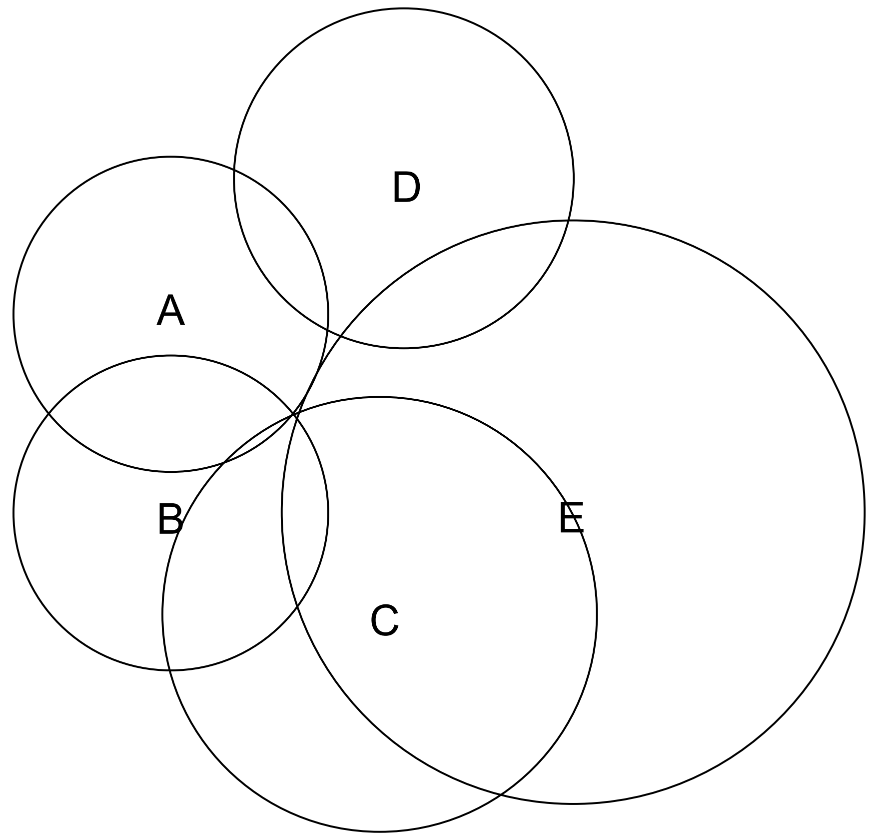
Effective Sound Velocity

- 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



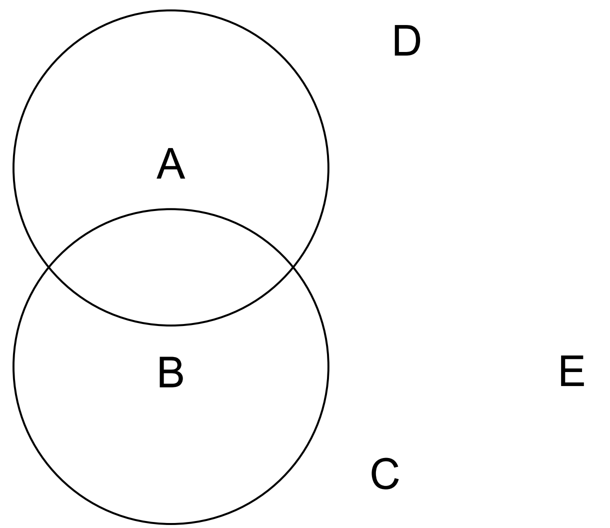
Spherical Tracking

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



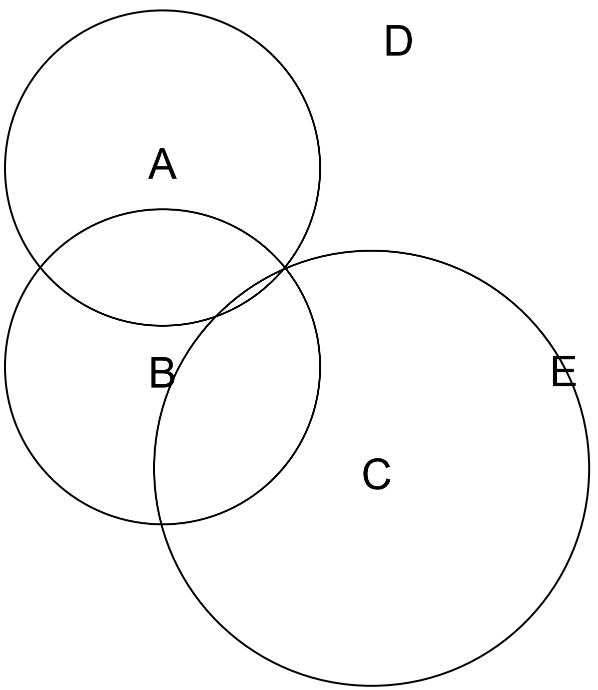
Spherical Tracking

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



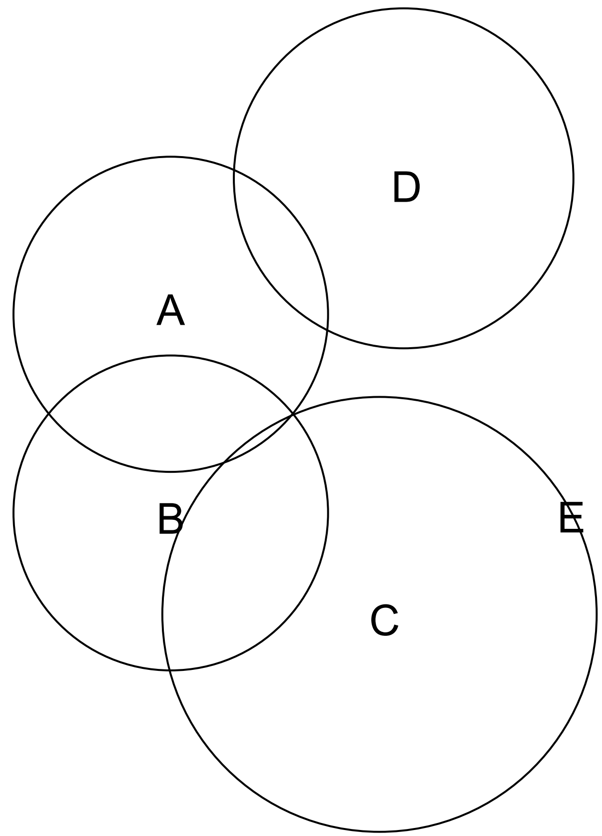
Spherical Tracking

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



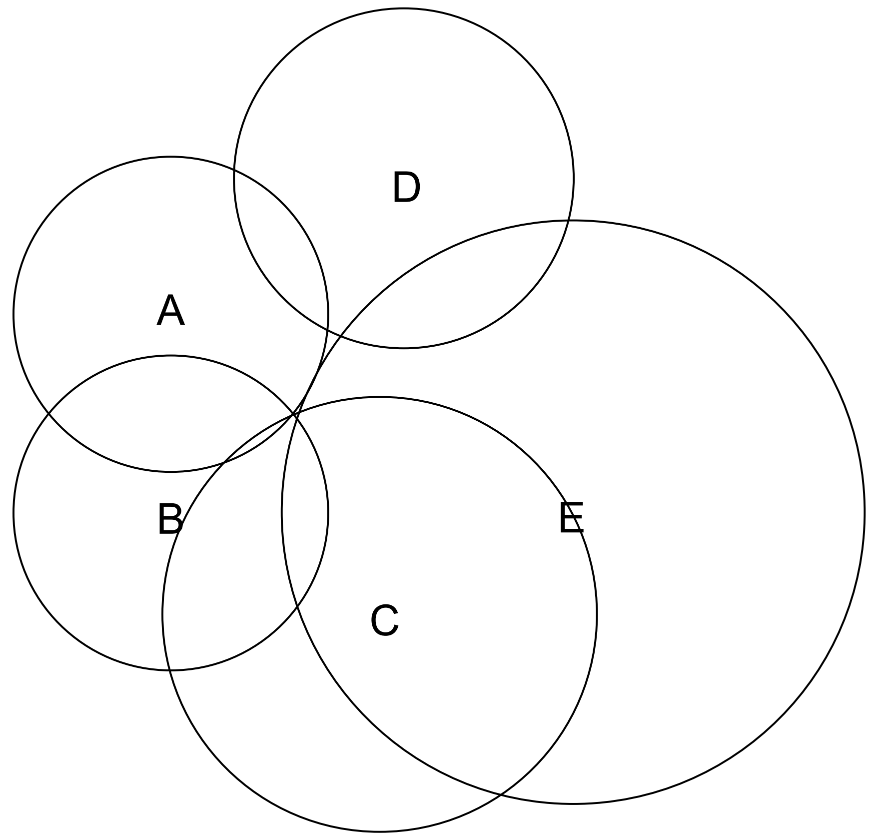
Spherical Tracking

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



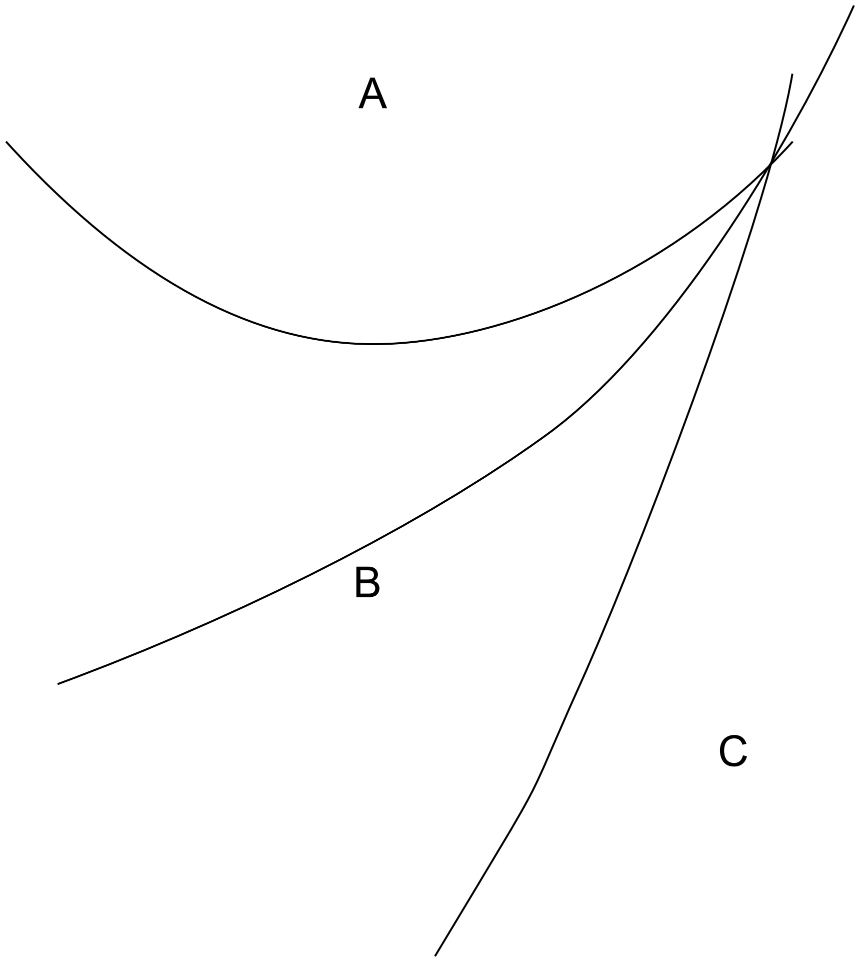
Spherical Tracking

- 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

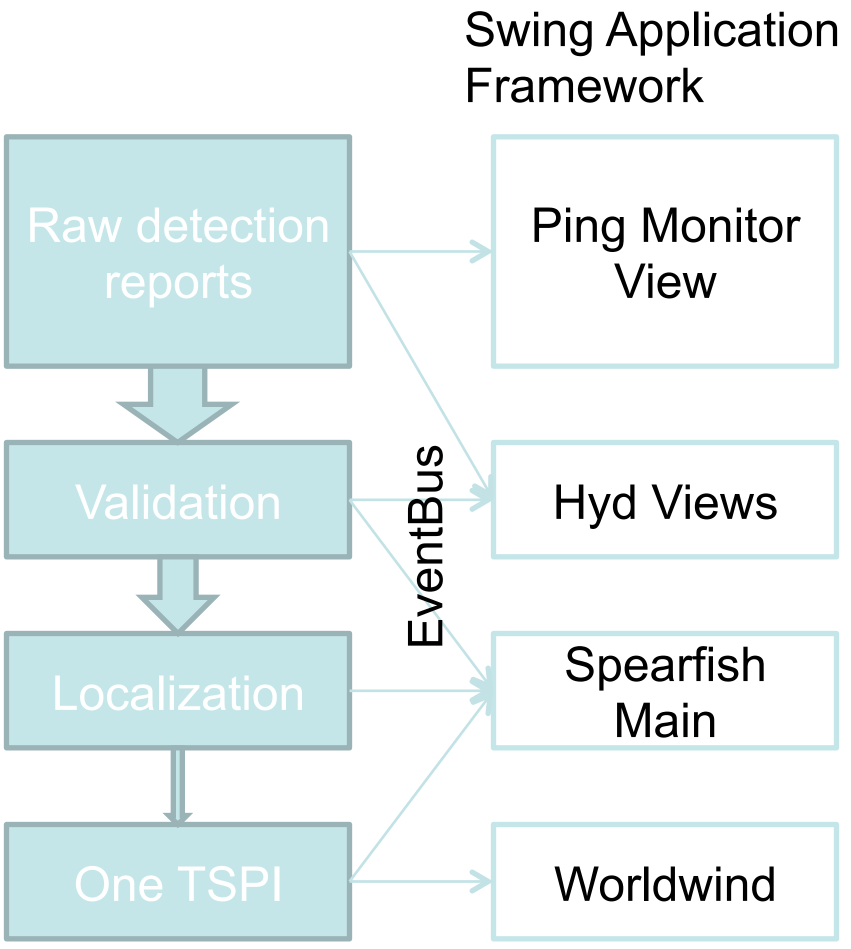


Standard Tracking Scenarios

- 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



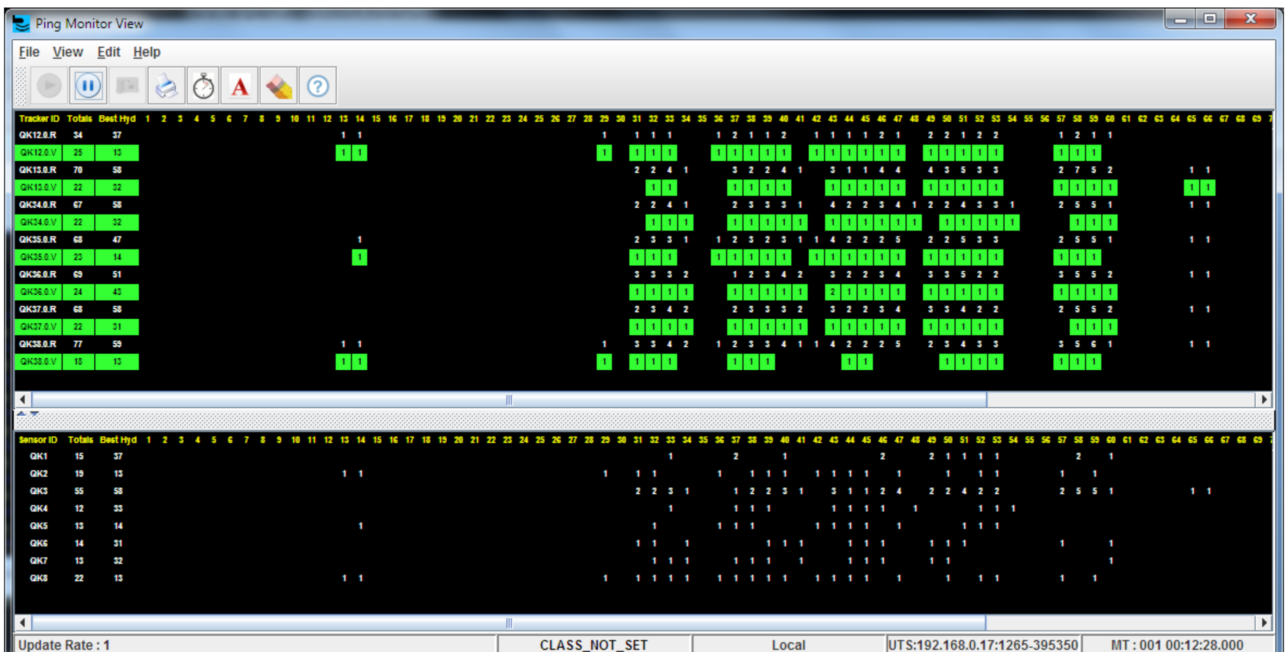
User Interface: Spearfish

- 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	1.00002	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	1.00001	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

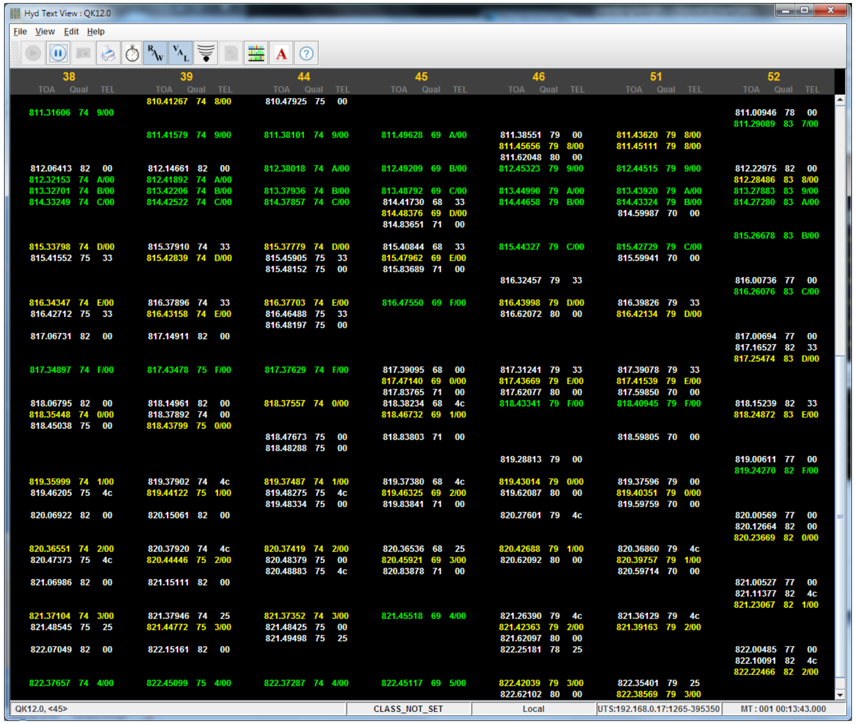
User Interface: Ping Monitor View

- 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

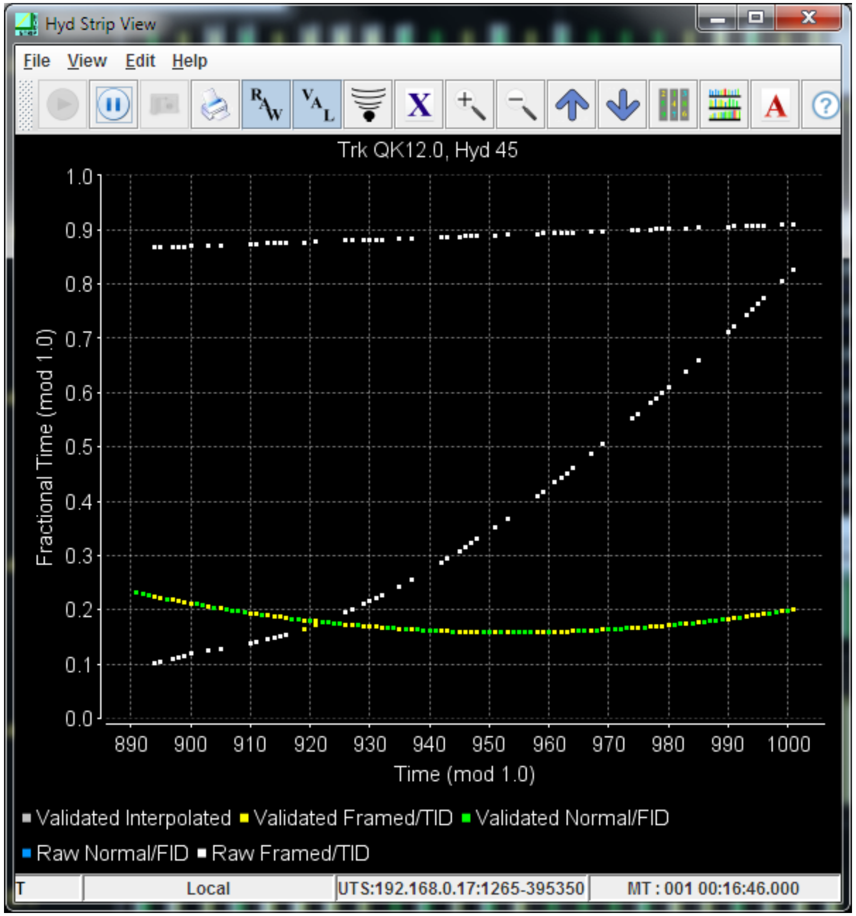


User Interface: HydGraphView

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

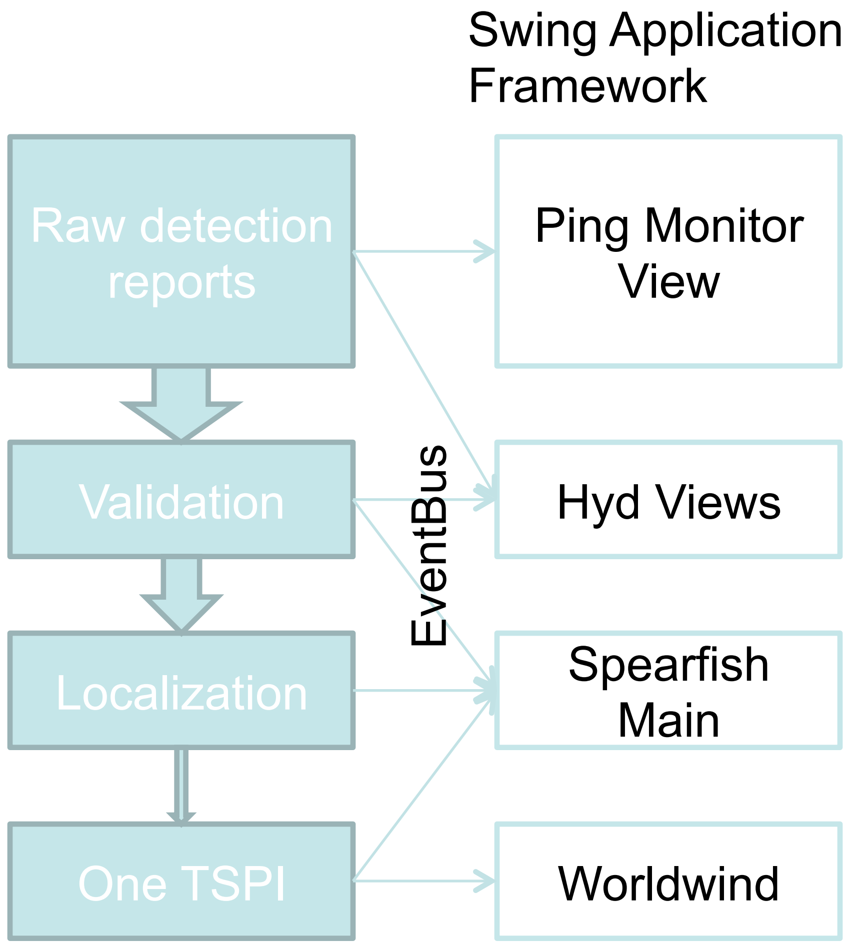
User Interface: HydStripView

- 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

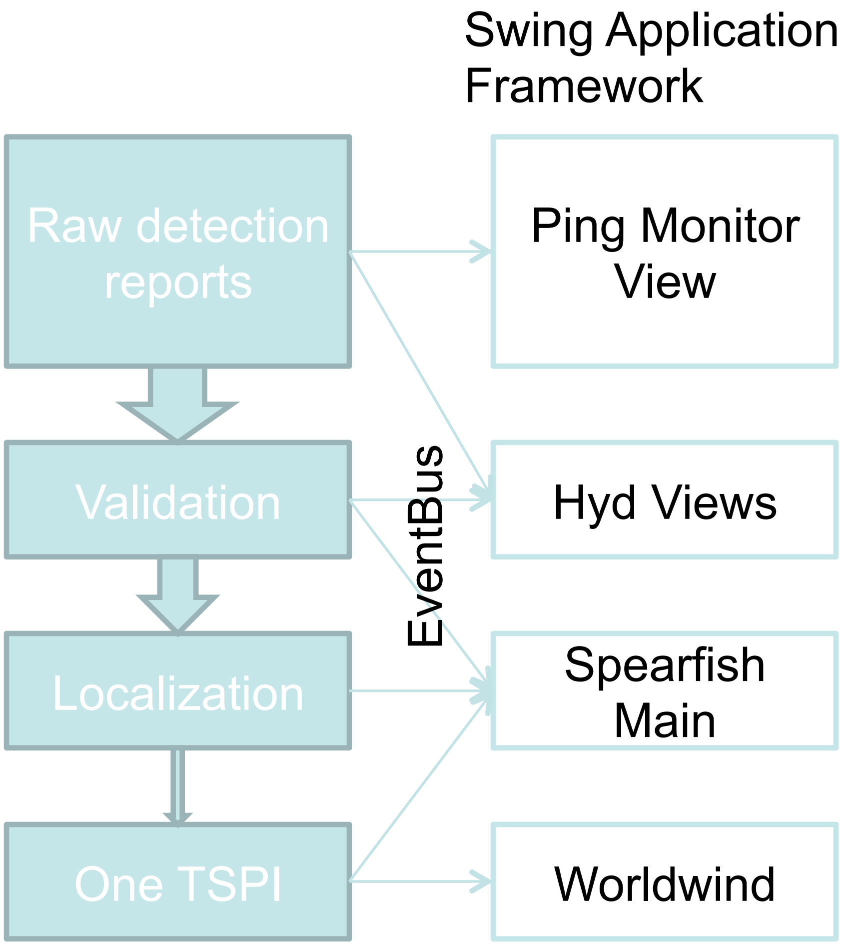
- Java 1.0 to 7
  - Initial work began in mid 90s
  - Under continuous development since
- Correctness reminder:
  - No data loss
  - GUI & processing should not interfere



Concurrency: RMI vs EventBus

- RMI = GUI and processing impact

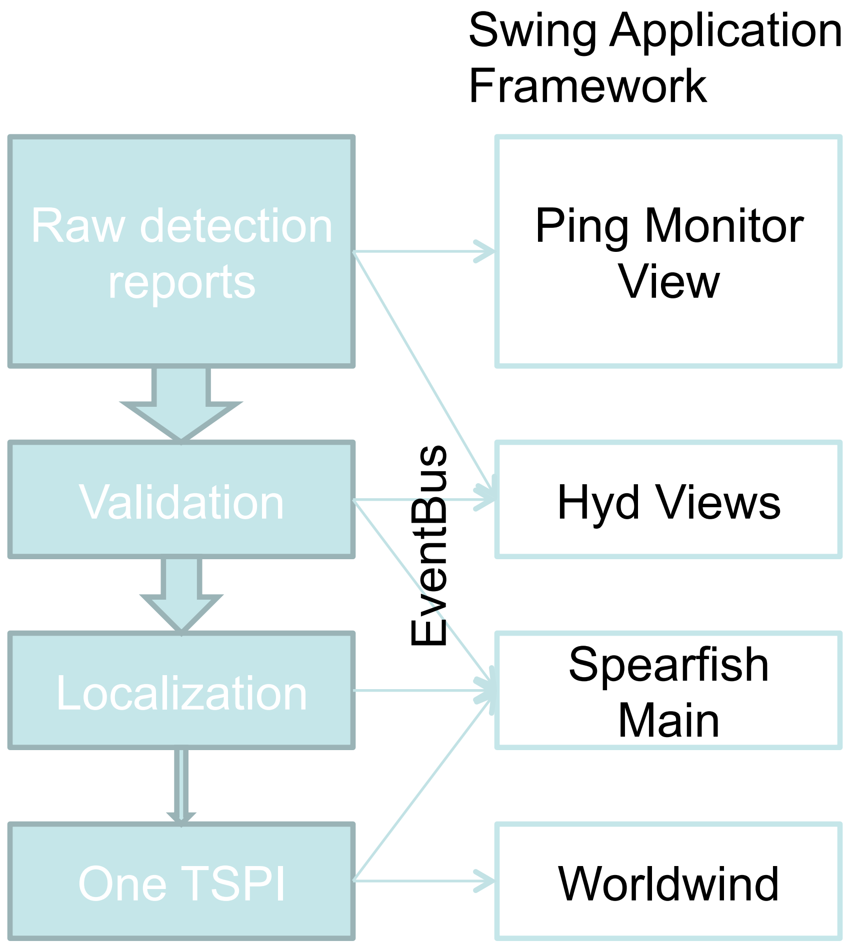
```
display() {
    // RMI call blocks GUI
    validation.getDR();
    // Processing and
    // Swing threads
    // coupled
    chart.showDR();
}
```



Concurrency: RMI vs EventBus

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

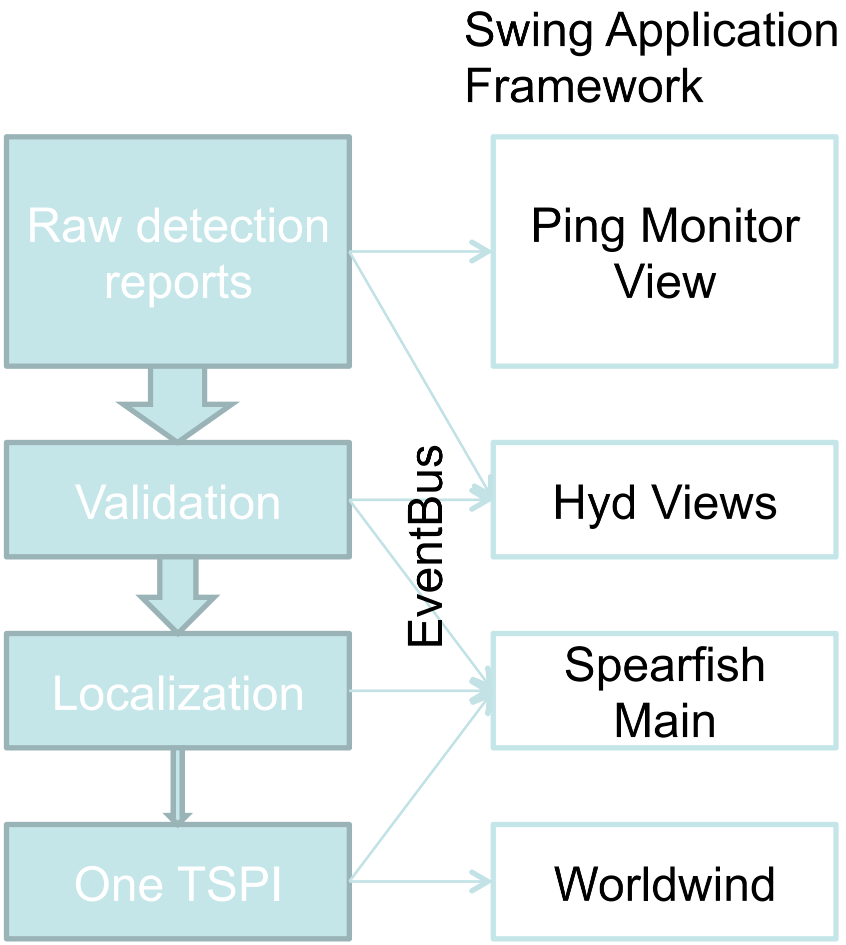
```
onEvent(dr) { // EB thread
    // Contention
    list.add(dr);
}
display() { // Swing thread
        // Contention
        timeSeries.add(list);
        chart.display(timeSeries);
}
```



Concurrency: CopyOnWrite

- CopyOnWrite
  - No corruption
  - No locking

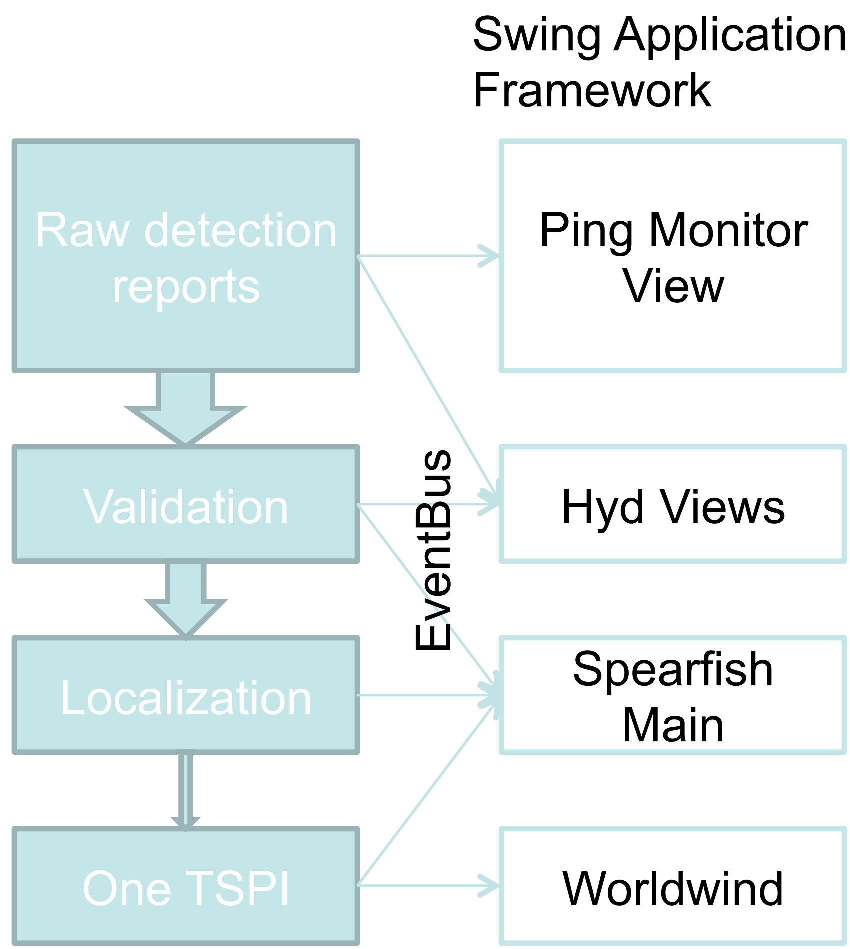
```
onEvent(dr) { // EB thread
    // No contention
    cowList.add(dr);
}
display() { // Swing thread
      // Diff copy from above
      timeSeries.add(cowList);
      chart.display(timeSeries);
}
```



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);
      });
}
```



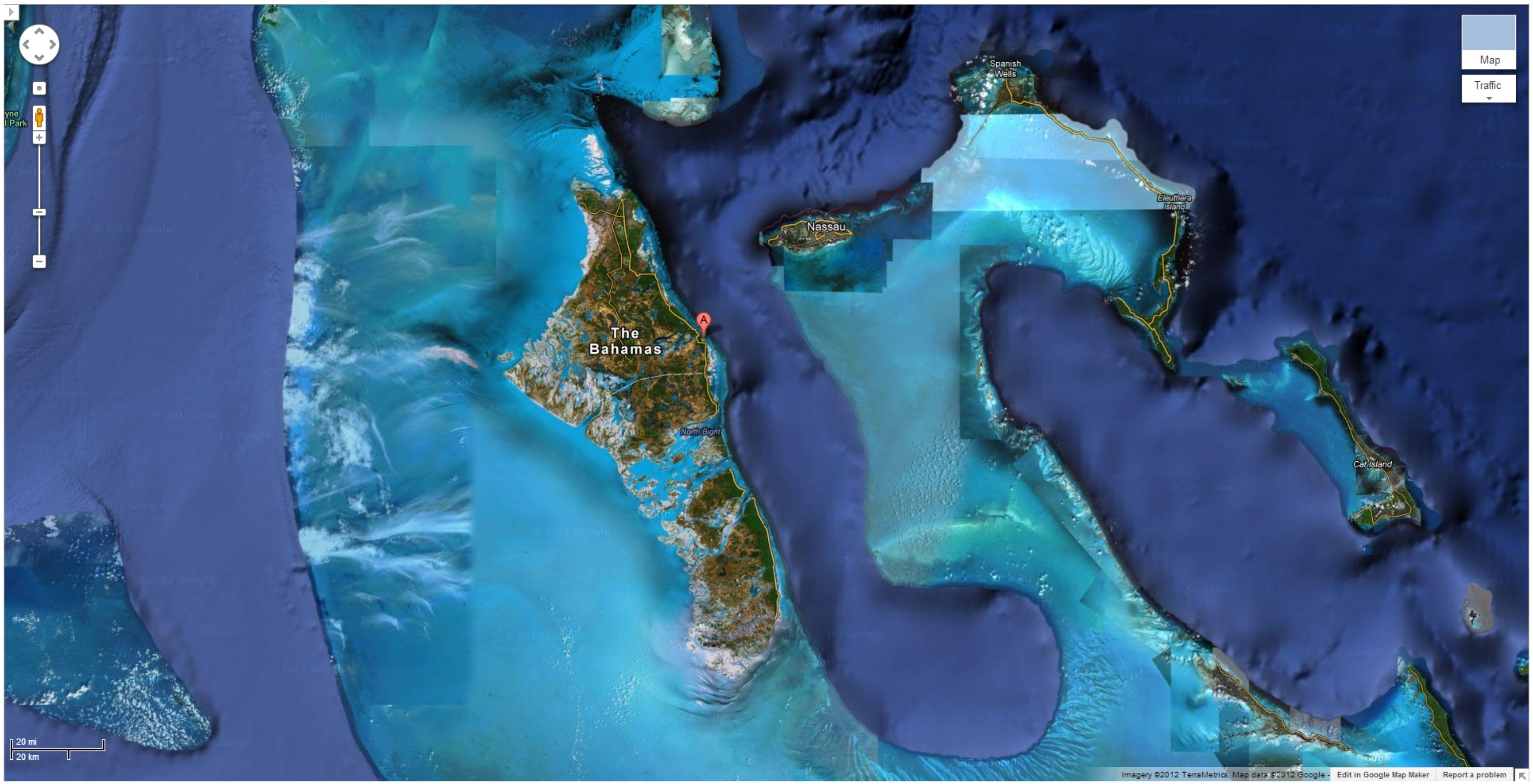
Concurrency: Basics?

- Why discuss relatively basic concurrency mechanisms?
- Large Java software systems’ concurrency correctness is 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


Map
Traffic
Spanish Wells
Nassau
Eleuthera Island
The Bahamas
North Bight
Cat Island
20 mi
20 km
Imagery ©2012 TerraMetrics, Map data ©2012 Google -
Edit in Google Map Maker
Report a problem

Bob Cross

Robert.a.cross1@navy.mil

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