GRID PROTECTION ALLIANCE

openHistorian 2

High-Performance Measurement Archive

GPA User's Forum 2015

Atlanta, Georgia

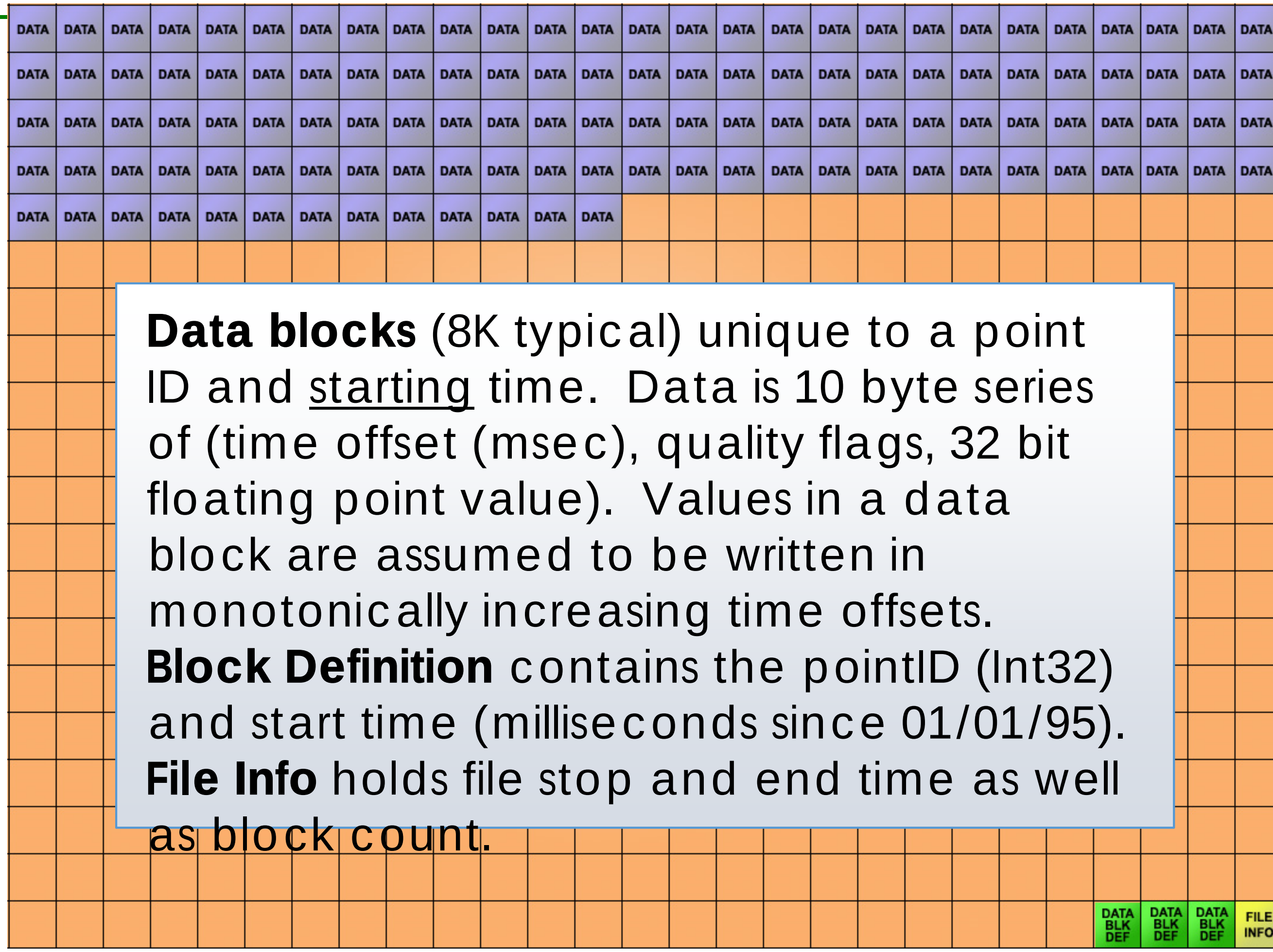
Version 1.0 - Major Deployments

- TVA
- Entergy
- PG&E
- Dominion
- Every openPDC installation (via stats and/or active phasor archive)

Version 1.0 - Current State

- Stable, mature product optimized to store time-series data
- Millisecond time-resolution
- 32-bit floating point values (with quality)
- Consumption limit around 100 PMUs
  (200,000 points per second per instance)
- 3x real-time replay speed
- Supports master/slave metadata modes
- GPA has deployed the historian with the openPDC -- no standalone installation
- File format identical to TVA DatAWare
  in use since mid-90s at TVA generation facilities

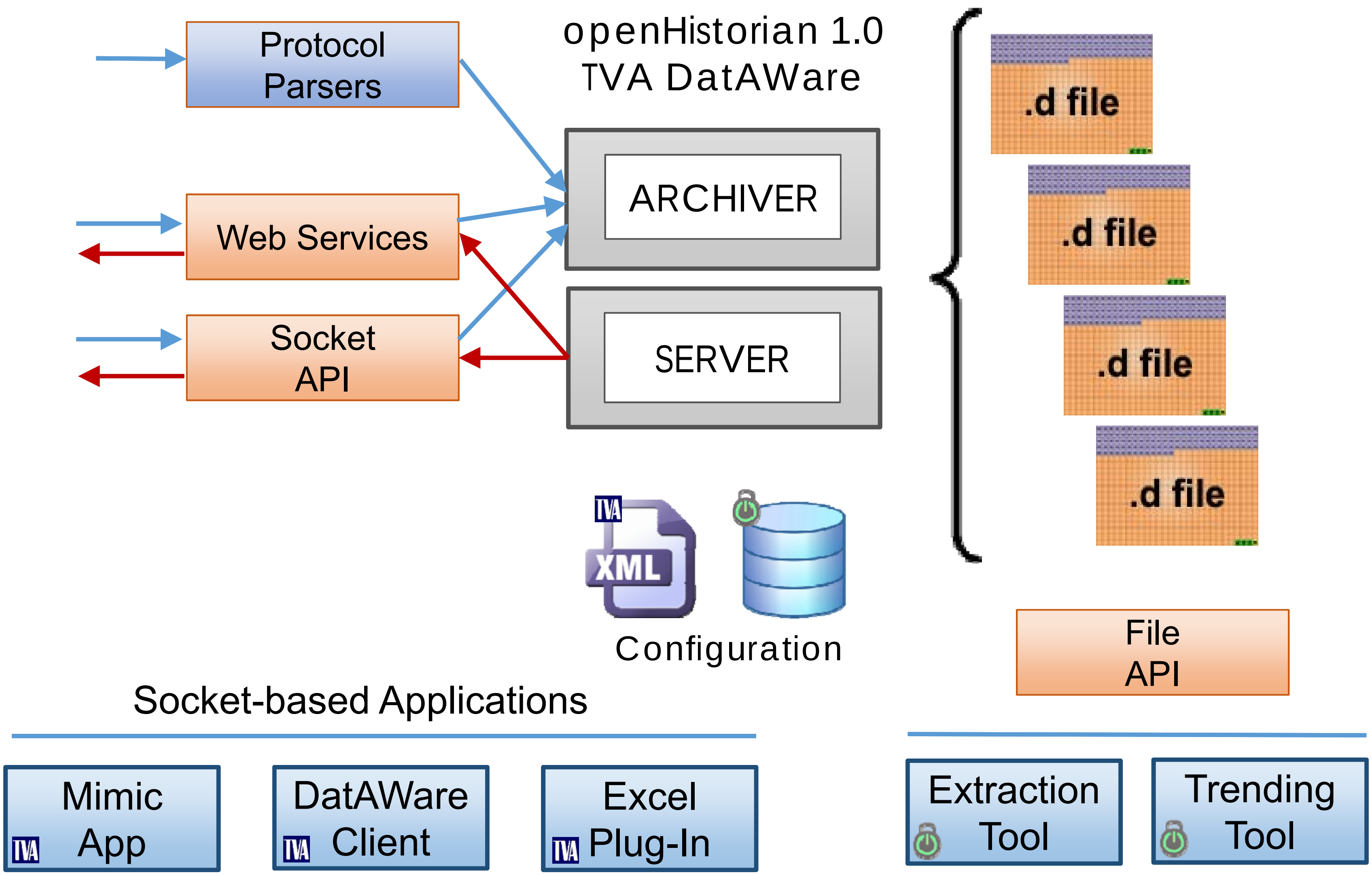
Version 1.0 File Format



Data blocks (8K typical) unique to a point ID and starting time. Data is 10 byte series of (time offset (msec), quality flags, 32 bit floating point value). Values in a data block are assumed to be written in monotonically increasing time offsets.
Block Definition contains the pointID (Int32) and start time (milliseconds since 01/01/95).
File Info holds file stop and end time as well as block count.

File sizes are typically about 1.5 GB.

System Components



GRID PROTECTION ALLIANCE

TVA Method to Distribute Data

- A prepositioned messaging service approach
  - A DLL is created and compiled into both the *open*Historian and all applications that consume this message.
- Message delivery is via socket connection using serialized data
- Data is pushed to the application (other methods used for request/reply, e.g., web services and the legacy socket API)
- Server side configuration of points and message structure to be distributed via encapsulated DLL with associated XML configuration file

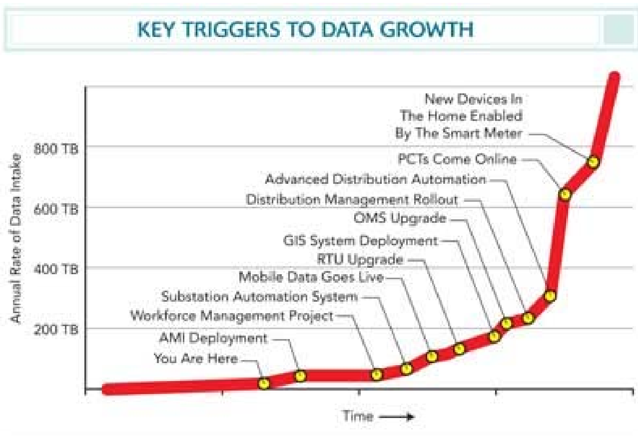
GRID PROTECTION ALLIANCE

openHistorian 2

High-Performance Measurement Archive

Case Study – Phasor Data

One of “Big Data” Sources from the Grid

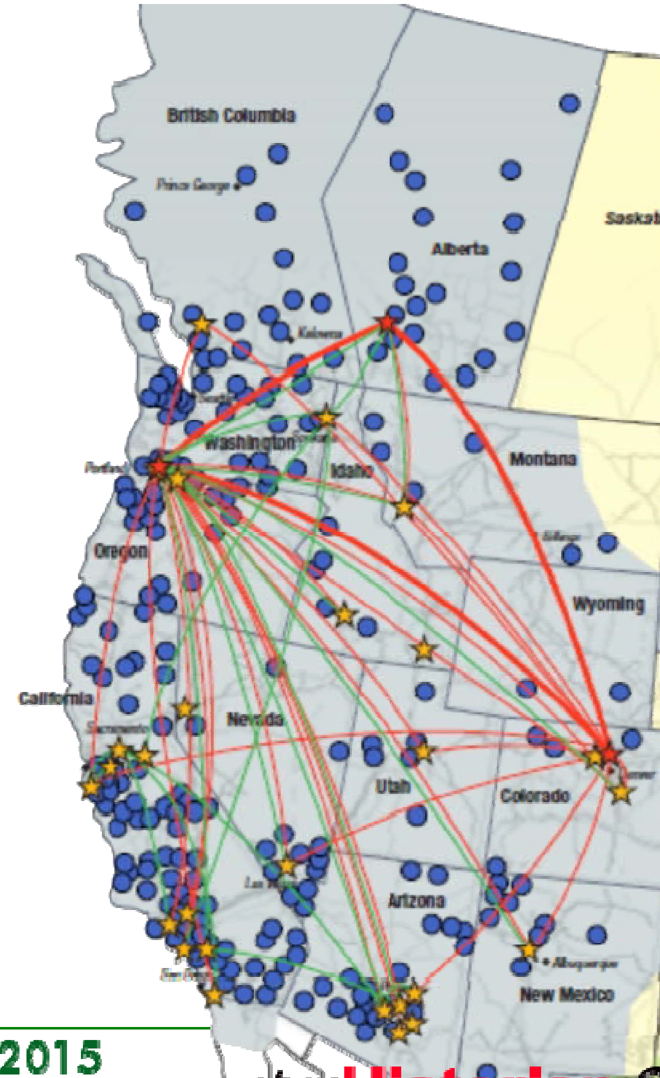


Grid data expected to grow rapidly in complexity and scale.

Company	Approx. PMU Count	Storage Technology
Large > 100 PMUs		
MISO	650	Oracle (150 PMUs Retained)
PJM	375	SQL Server
OG&E	250	SQL Server
WECC	170	OSIsoft-PI
BPA	130	OSIsoft-Pi (most at 60 sps)
Duke	100	OSIsoft-PI
Medium > 25 PMUs (1 GB / Day or less)		
Dominion	80	openHistorian
ISO-NE	80	openHistorian/PhasorPoint
CALISO	60	OSIsoft-PI
Entergy	50	OSIsoft-PI / openHistorian
FP&L	50	eDNA
NYISO	50	OSIsoft-PI
ERCOT	45	OSIsoft-PI
PG&E	40	OSIsoft-PI / openHistorian
Plus Others		
Small < 25 PMUs		
TVA	20	openHistorian
Plus Dozen's of Others		

WISP Project Lessons Learned

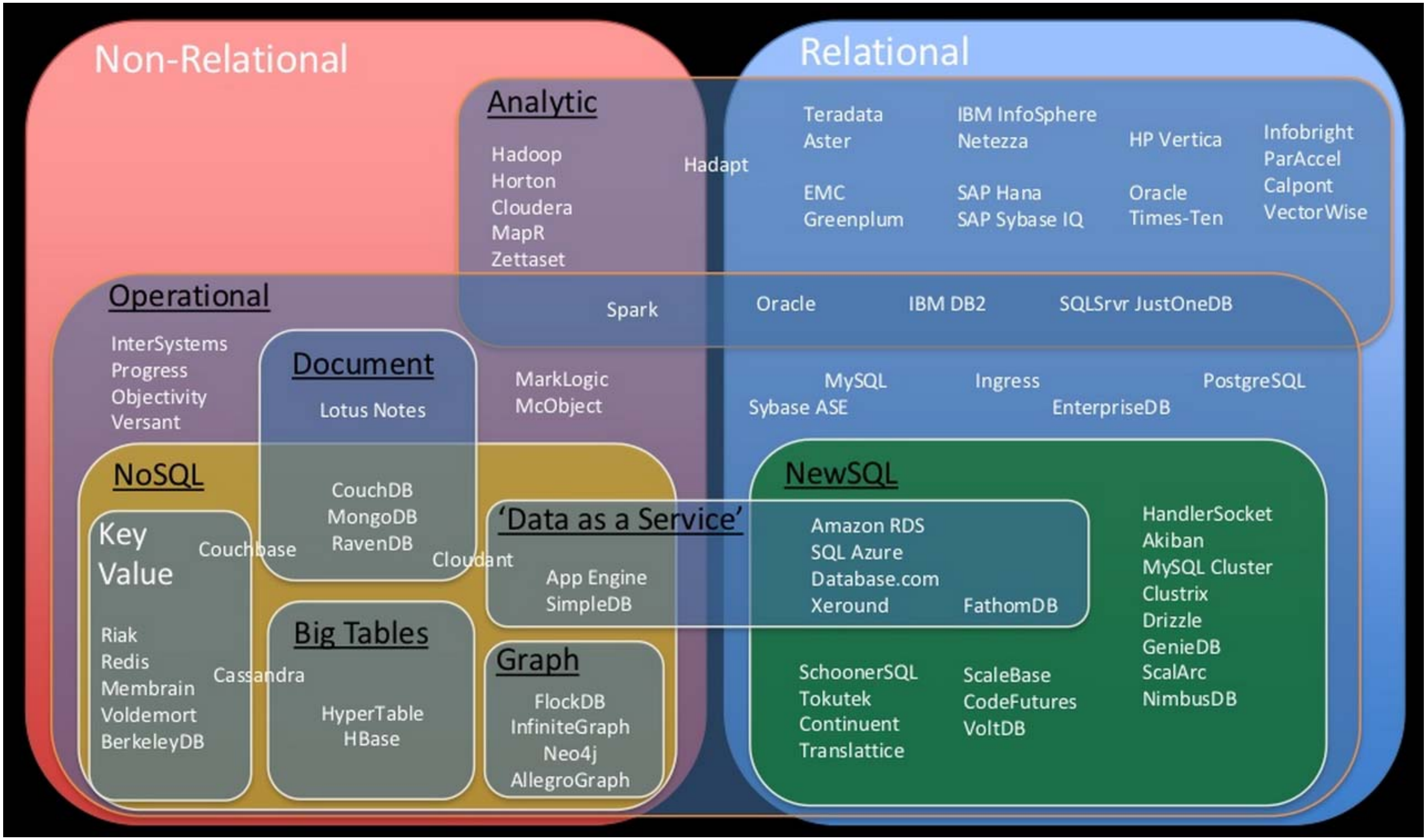
- Availability and accuracy of the data
- Data mining tools for information extraction
- Difficulty in deploying a common naming convention
- Upgrading first releases of vendor products to CIP security standards
- Applications unproven (finding and working out the bugs)
- Integrating old PMUs and PDCs
- **Applications stressed by large data volumes**



A new data layer must support:

- High performance processing of time-series data
  - For both data archiving and retrieval modes
  - High frame rate application refresh / Quick app response time
  - Fast extraction of large data block – e.g., a day’s data
- An expanded set data types (e.g., doubles, strings, complex values, etc.) while maintaining low storage requirements
- GPS precision time stamping
- Ability to insert data out of sequence
- Lossless data compression
- Improved Interfaces
  - High-speed socket-based API for data access
  - GEP based pub/sub real-time data subscription

Big Data Problem – One Size Doesn't Fill All



Source: Infochimps, May 2012

The Solution – GPA's SNAPdb Library

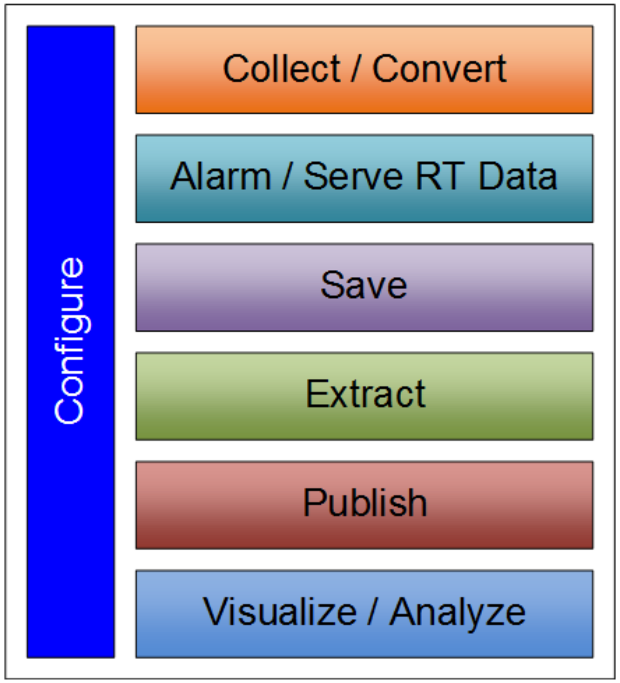
- Serialized
- NoSQL
- ACID Compliant
- Performant

- Housed within GPA's
  Grid Solutions Framework

What is ACID?

- SNAPdb Implements ACID to protect data integrity.
  - **Atomicity** - requires that database modifications must follow an "all or nothing" rule. Each transaction is said to be atomic
  - **Consistency** - ensures that any transaction the database performs will take it from one consistent state to another
  - **Isolation** - refers to the requirement that no transaction should be able to interfere with another transaction at all
  - **Durability** - that once a transaction has been committed, it will remain so

*open*Historian 2.0 Components



API allows adapters to be developed for data inputs

Recent data served with very low latency from memory

High performance data storage with lossless data compression

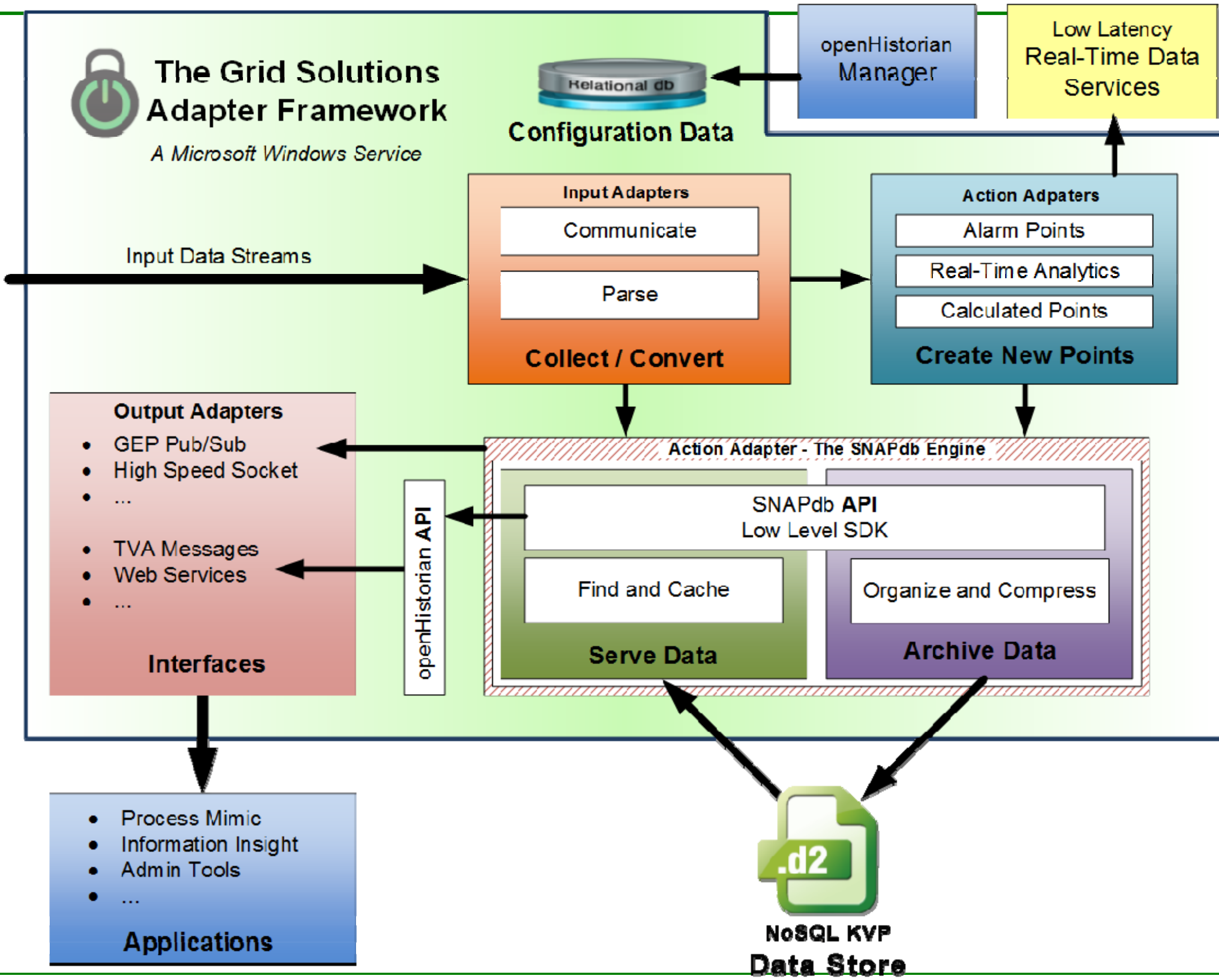
Tools provided for historical data extraction

Both publish / subscribe and web services interfaces

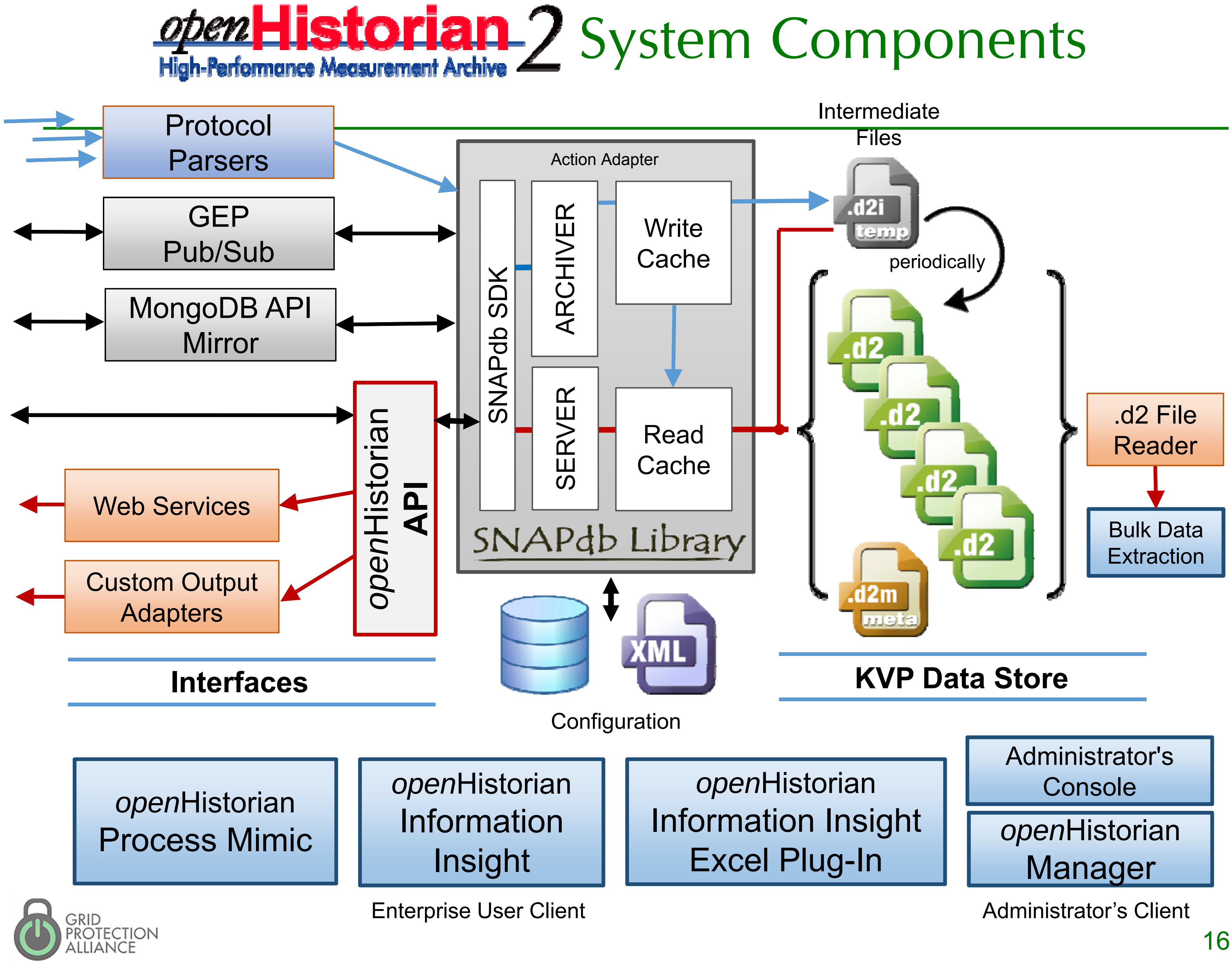
Planned companion tools allow users to view and export data

Configuration is integrated across *open*Historian processes and can itself be integrated other configuration information data sources.

The openHistorian Leverages the GSF



openHistorian 2 System Components



SNAPdb Data Structure

- B+Tree based that supports out of sequence insertion
- Time support to +/- 100 nanoseconds (ticks)
  (with extended time precision fields available)
- Data can be of any type that can fit in 192-bits
  (for example, float32, float64, complex32, complex64, int32, int64, uint32, uint64, char and string16)
- Data stored sequentially in compressed 4K structures
- Tested and optimized for phasor data

SNAPdb's Key-Value Pair

- Key is a join of PointID and Time
- Value can be up to 192-bits*

Example format for the openHistorian:

Key				Value		
Ticks**	Point ID	Ext Prec.	Time Flags	DATA		Quality Flags
64 bits	64 bits	64 bits		64 bits	64 bits	64 bits

10 Bytes V1
48 Bytes V2
(unencoded)

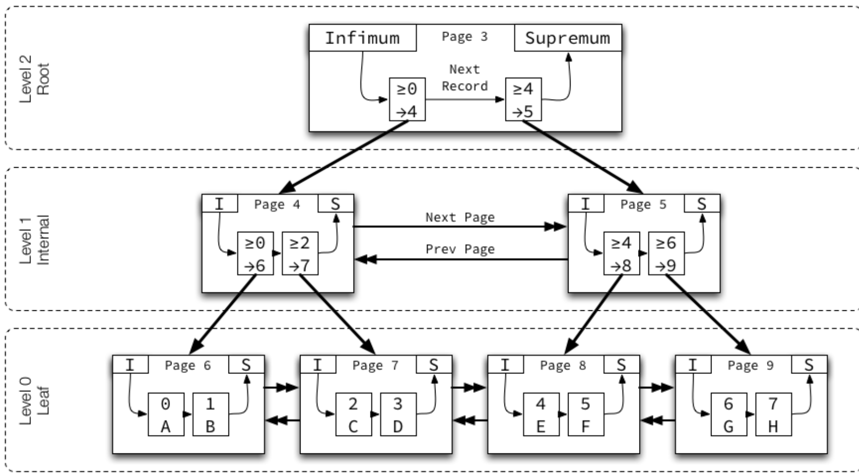
* 192-bits is used by the openHistorian as the size of values and keys – this is not a restriction of the SNAPdb.

**1 Tick = 100 nanoseconds
Time Flags = duplicate entry counter (for DST), leap second, etc.

The 64-bit PointID is normally not referenced at the user level. Rather a GUID is assigned
through configuration as well as primary and alias Tags.

B + Tree Overview

- Tree grows from the bottom up
- Leaf-nodes contain blocks of sequential data
- Nodes are doubly linked and point to previous and next node



- Tree indices are unsigned 32-bit integers which require internal-nodes to support large trees

SNAPdb's B+Tree

- The leaf node of data is encoded and decoded dynamically to optimize memory and storage
- Implementation operates directly from disk without requiring an in-memory data structure
- The data encoding process is used to implement lossless data compression

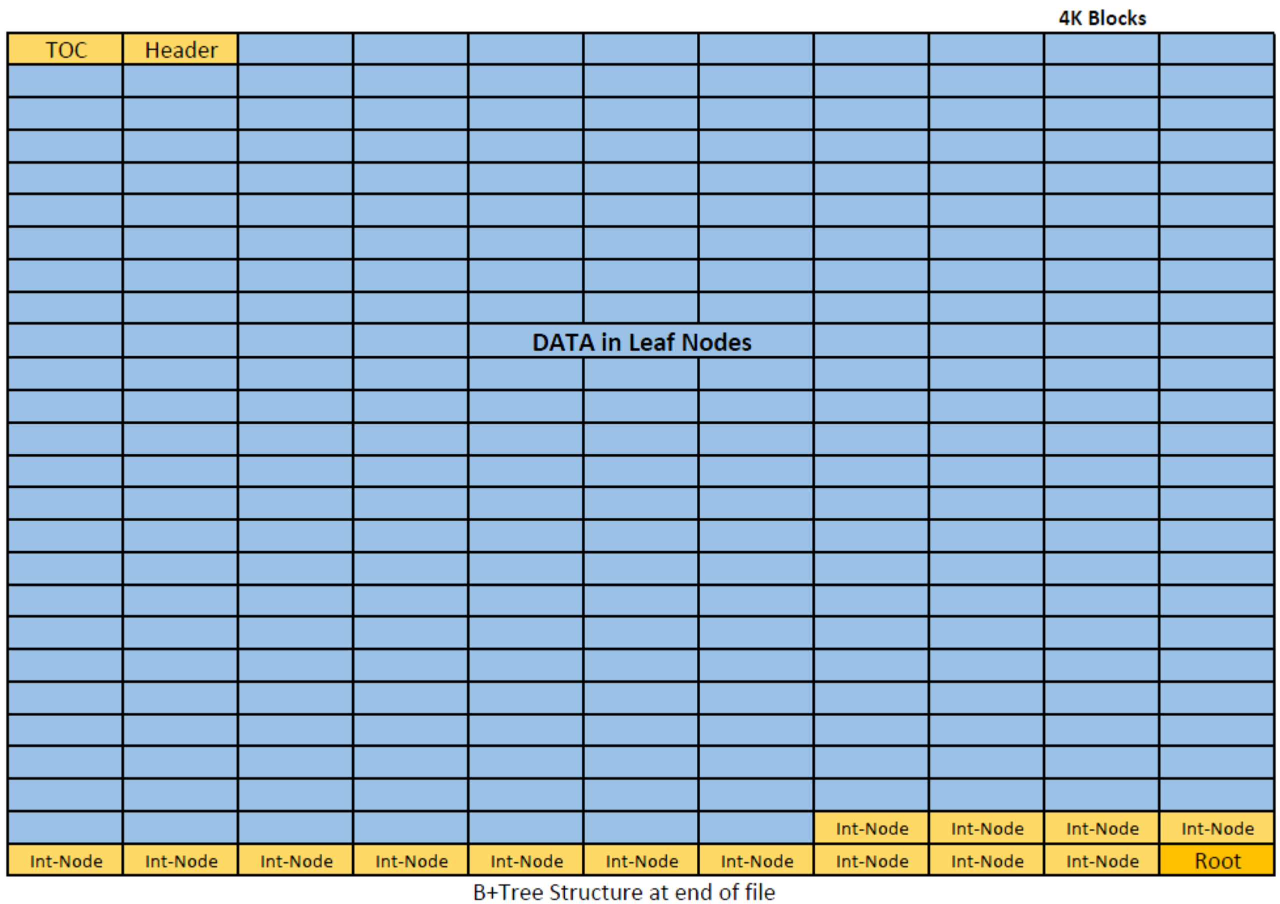
.d2 File Format

- The .d2 file contains a table of contents, a B+Tree header and nodes: a root node, internal nodes and leaf nodes

Node Structure

Version #	Node Level	Record Count	Bytes Used	Left Sibling
1 Byte	1 Byte	2 Bytes	2 Bytes	4 Bytes
Right Sibling	**Lower Key**	**Upper Key**	**Data Block**	**Footer Block**
4 Bytes	24 Bytes	24 Bytes	4000+ Bytes	32 Bytes

The .d2 file Data Block is a B+Tree Collection of 4K Leaf Nodes



.d2 File Creation Process

LOCAL RESOURCES

- **Real-time –** Recent data cache
- **Stage 1** – Flushes real-time cache to disk
  (10 second default)
- **Stage 2** – Consolidates Stage 1 files
  (created based on either size or time constraints)

SHARED

- **Stage 3** – Create final .d2 archive file
  (created based on size constraints)

  Recommended size 2GB

System Metadata Files

- Enables the .d2 files to be independent of master configuration systems
- Association of .d2 and .d2m files provides meta-data versioning over time
- Associates internal openHistorian 2.0 key (a long integer) with its configuration GUID
- Contains the value's fundamental meta data
  - Data Type
  - Measurement Units
  - Preferred Tag
  - Short Description

GSF Input Adapters / Protocol Parsers

- GSF input protocol parsers are all included with the openHistorian as part of the SNAPdb's integration with the framework
- The supported protocols include:
  - DNP3
  - IEEE C37.118
  - IEC 61850-90-5
  - CSV
  - OSI-PI
  - IEEE 1344
  - BPA PDCstream
  - SEL FastMessage
  - UT F-NET
  - COMTRADE

Summary – Version 2.0 Improvements

- Fast
  - In-memory cache for very high speed extraction of near-real time data
  - Low data insertion lag time
  - High-speed API for historical data extraction
- Reliable
  - ACID-based system design objectives
    -- with emphasis on "durability"
  - File structure resistant to data corruption
- Expanded Use Capability
  - Out-of-time-sequence inserts allowed
  - Transaction-like data updates allowed
  - Loss-less data compression
  - More data types
  - Better interfaces

Version 2.0 - Current State

- Early Beta version released
  (and in pre-production use at OG&E, TVA and by UT's CURENT center)
- Testing, Bug Fixes and Benchmarking in progress
- Source code available from codeplex:

http://openhistorian.codeplex.com

openHistorian API

- Archive and read support for data – historical and real-time with low latency – for point selection over time-range
- Updates and deletes could be implemented – but are purposely not enabled for the historian use case
- Multiple data types
- Socket or local files implementations
- Interval based data retrieval options – enables high-speed data zooming
- Server-side data filtering

Data Read API

```
// Example:
var enumerator = GetHistorianData("127.0.0.1", "PPA",
DateTime.UtcNow.AddMinutes(-1.0D), DateTime.UtcNow)

// API:
IEnumerable<HistorianMeasurement> GetHistorianData(
      string historianServer,
      string instanceName,
      DateTime startTime,
      DateTime stopTime,
      string measurementIDs = null)
```

Data Write API

```
// Example:
WriteHistorianData("127.0.0.1", "PPA", measurements)

// API:
void WriteHistorianData(
        string historianServer,
        string instanceName,
        IEnumerable<HistorianMeasurement> measurements)
```

Also – SQL Server Adapter

- Can query trending data from within SQL Server using SQL CLR adapter:

SQLQuery11.sql - (local)\...\Ritc...(52))*

```
SELECT * FROM GetHistorianData(
        '127.0.0.1',
        'PPA',
        '2015-05-01 00:00:00',
        '2015-05-01 00:00:10',
        '1,2,3')
```

Results | Messages

	ChannelID	Time	Value
1	1	2015-05-01 00:00:00.0000000	8228
2	2	2015-05-01 00:00:00.0000000	59.97
3	3	2015-05-01 00:00:00.0000000	0
4	1	2015-05-01 00:00:00.0330000	8228
5	2	2015-05-01 00:00:00.0330000	59.962
6	3	2015-05-01 00:00:00.0330000	0
7	1	2015-05-01 00:00:00.0660000	8228
8	2	2015-05-01 00:00:00.0660000	59.968
9	3	2015-05-01 00:00:00.0660000	0
10	1	2015-05-01 00:00:00.1000000	8228
11	2	2015-05-01 00:00:00.1000000	59.957