A Case for using EPICSv4 in areaDetector

Bruno Martins

U.S. DEPARTMENT OF ENERGY

BROOKHAVEN NATIONAL LABORATORY

A few interesting EPICSv4 features

- pvData provides structured types
- Scalar types: bool, [u]int[8,16,32,64], float, double, string
- Normative Types: NTNDArray
  - Standardized
- Smart pointers throughout

Why use NTNDArray?

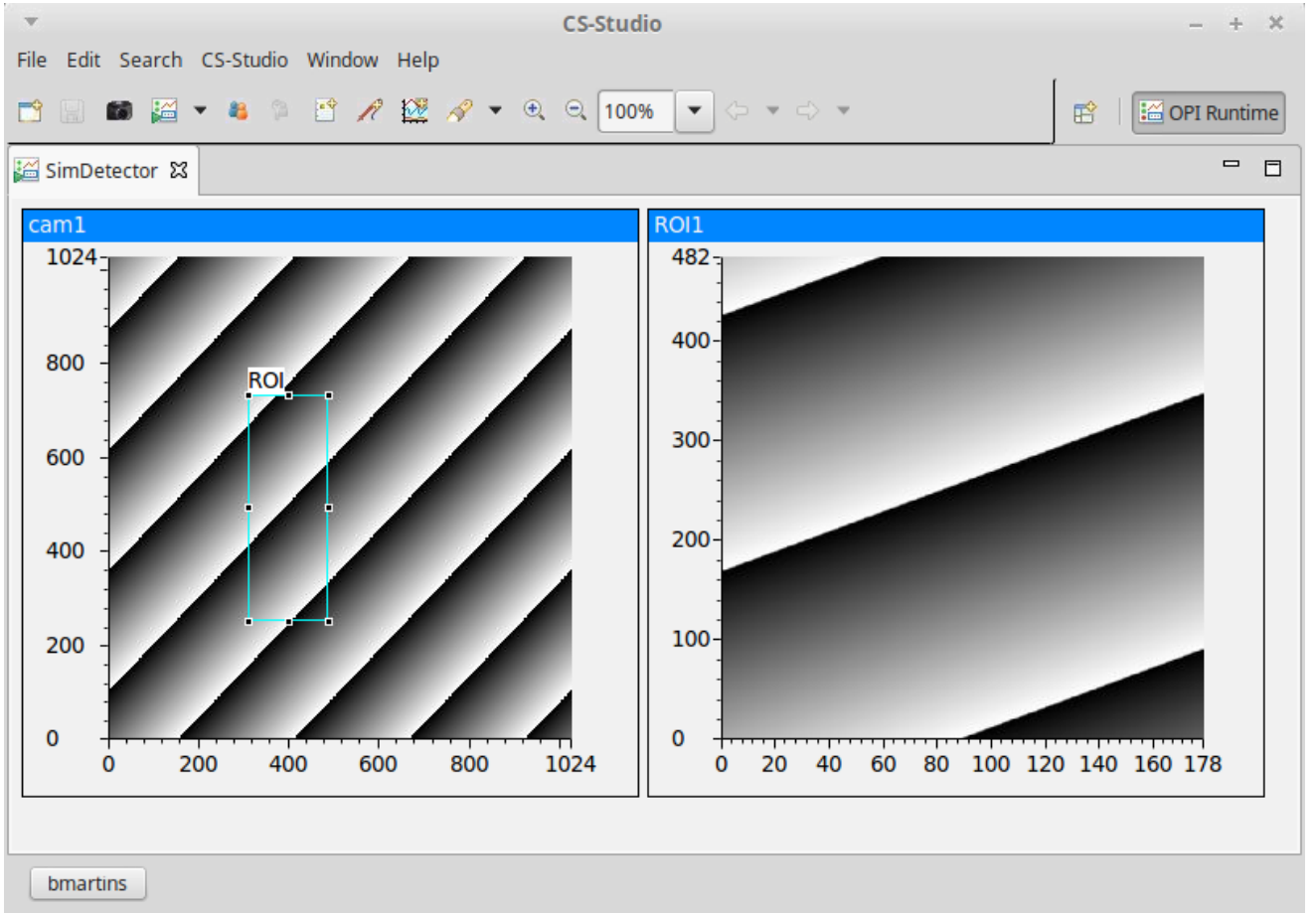
- Well defined normative type, modeled after areaDetector's NDArray
- A structured record rather than a set of separate records
  - Avoids conversion from NDArray object into waveform + ai/ao/... records
  - Same structure internally and on the wire: easy conversion

Why use NTNDArray?

- Allows for distributed processing
  - Multiple instances of areaDetector
- Zero copy if client and server are on the same machine
  - Done transparently

An Implementation

In order to explore some concepts, I did a test implementation: ADCore + simDetector + NDROI



An Implementation

- New NDArray
  - Wraps NTNDArray
  - Hides internal data
  - pArray->getData() rather than pArray->pData
  - Passed around as shared_ptr: NDArrayPtr

An Implementation

- New NDArrayPool
  - Doles out NDArrayPtr's
  - Automatic reference counting done by smart pointer
  - Automatic reclaims NDArray when not used anymore
    - No reserve() and release()

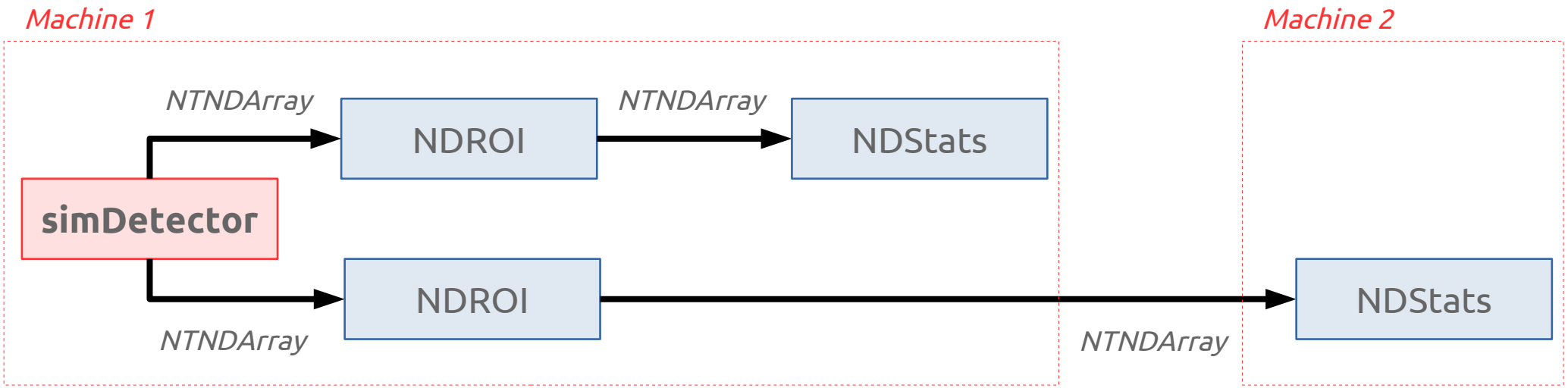
An Implementation

- Producers (drivers and plugins) serve NTNDArrays
- Consumers (typically plugins) monitor NTNDArrays
  - Since NTNDArray is the backing structure for the new NDArray, “conversion” is straightforward
- Transparent zero copy on the same machine

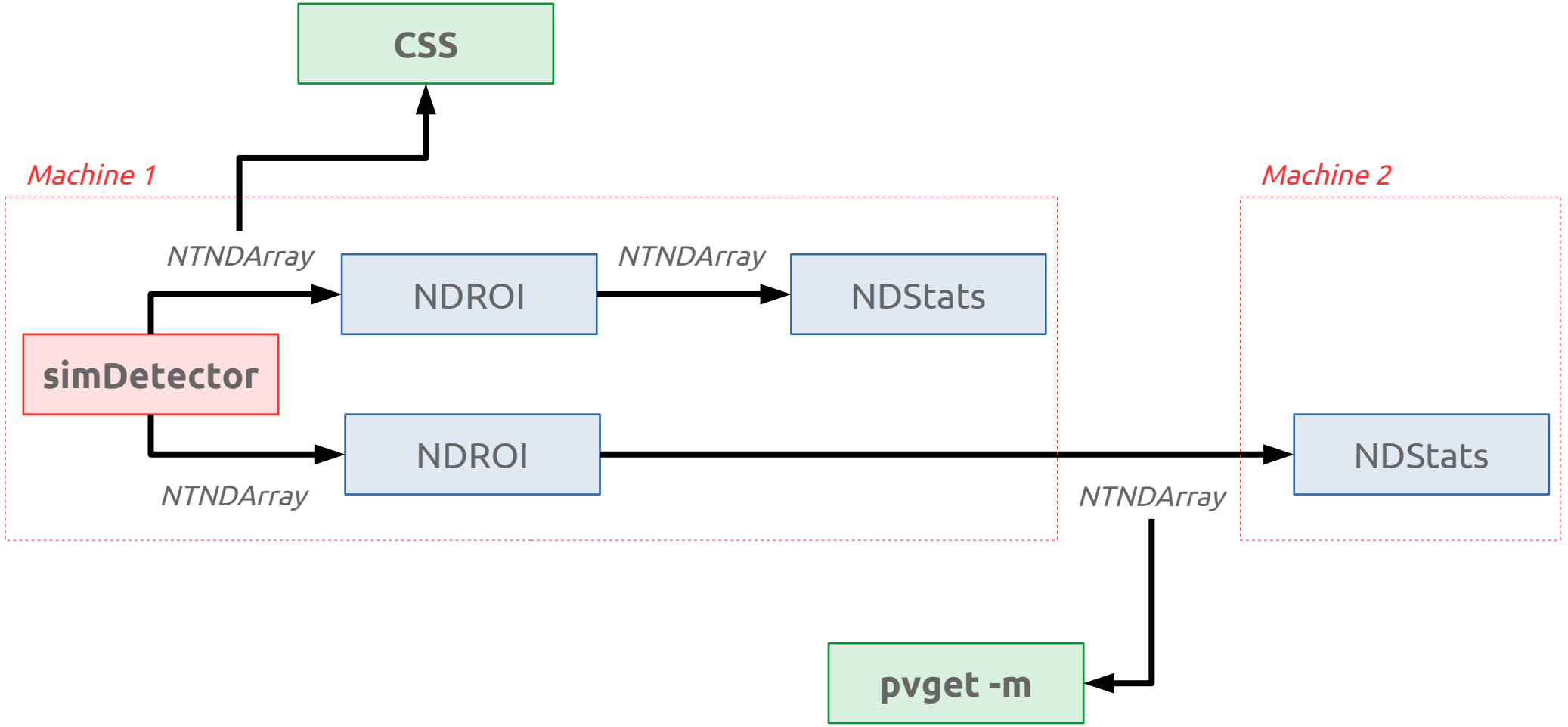
ADDriver

NDPlugin

Plugins are clients (monitors)



Other clients can monitor any array



Why stop there?

Why stop there?

- Right now areaDetector drivers and plugins are built on top of asynPortDriver
  - The parameter library allows for int32, float64 and string parameters

- Why not build a parameter library with pvData / pvDatabase?

Why stop there?

- If used, NTNDArray would introduce a dependency on pvData, pvAccess and normativeTypes anyway
  - Until they are merged into EPICS base

- A new parameter library would be able to provide scalar parameters of every supported scalar type:
  - bool, [u]int[8,16,32,64], float, double, string
  - New dependency on pvDatabase

Why stop there?

- Actually, parameters could be any structure supported by pvData!
  - An NDArray would be just another parameter, not special at all

An Implementation: pvPortDriver

- A replacement for asynPortDriver
- Every parameter becomes an entry in the underlying pvDatabase
- Parameters have a prefix, passed at pvPortDriver's construction time

Creating and using parameters

```
// Header File
class ADDriver {
    epics::pvPortDriver::StringParamPtr ADManufacturer;
```

Parameter type is known at compile time, not an index into a table

```
// Inside ADDriver constructor
ADManufacturer = createParam<string>("Manufacturer_RBV");
```

A PV is created. For example: 13SIM1:cam1:Manufacturer_RBV

```
// Using the parameter
ADManufacturer->put("Some Manufacturer");
```

All monitors will be notified

```
string manufacturer = ADManufacturer->get();
```

Value is returned directly

Creating and using parameters

```
// Header File
class ADDriver {
    epics::pvPortDriver::EnumParamPtr ADStatus;
}


// Inside ADDriver constructor;
vector<string> ADStatusChoices(ADStatusAborted+1);
ADStatusChoices[ADStatusIdle]    = “Idle”;
ADStatusChoices[ADStatusAcquire] = “Acquire”;
//...
ADStatusChoices[ADStatusAborted] = “Aborted”;
ADStatus = createEnumParam(“ADStatus”, ADStatusChoices);


// Using the parameter
ADStatus_t status = (ADStatus_t) ADStatus->get();
```

Downsides

- Asyn is used everywhere, is well established and well debugged; new code always introduces new bugs
- Reimplementing all of Asyn's functionalities will take a ***lot*** of work
- If asynPortDriver is replaced, there's no devEpics equivalent
  - No immediate support for V3 databases

Thank you!