What's with all the 1s and 0s?

Making sense of binary data at scale with Apache Tika

APACHECON NORTH AMERICA

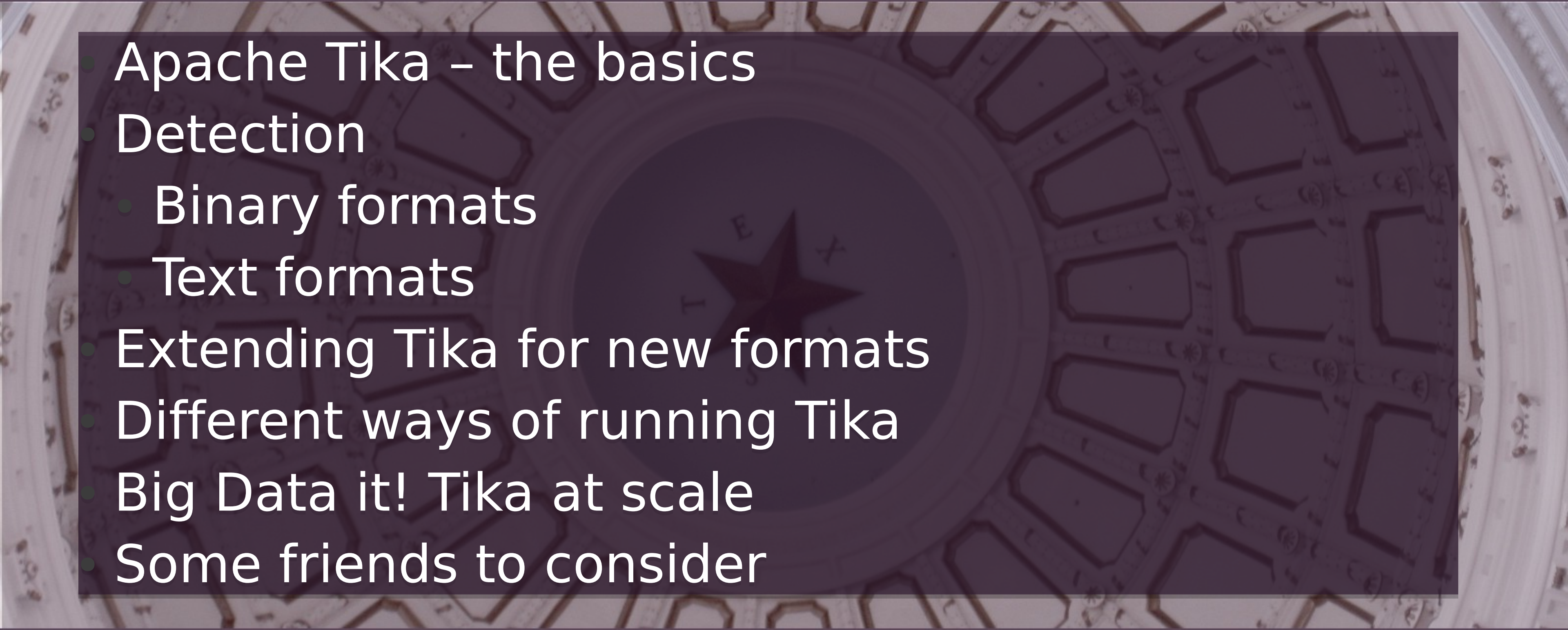
Nick Burch

CTO, Quanticate

Quanticate

APACHECON
NORTH AMERICA

Those 1s and 0s

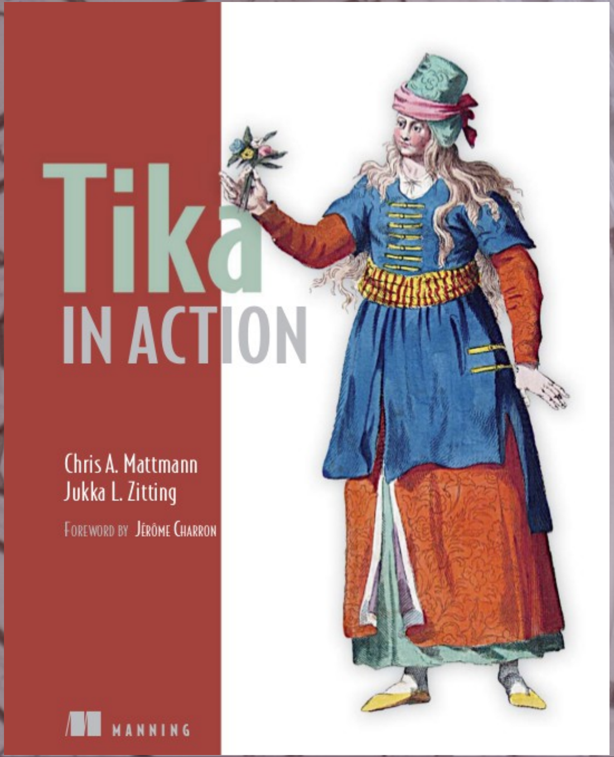


- Apache Tika – the basics
- Detection
  - Binary formats
  - Text formats
- Extending Tika for new formats
- Different ways of running Tika
- Big Data it! Tika at scale
- Some friends to consider

Tika in a nutshell

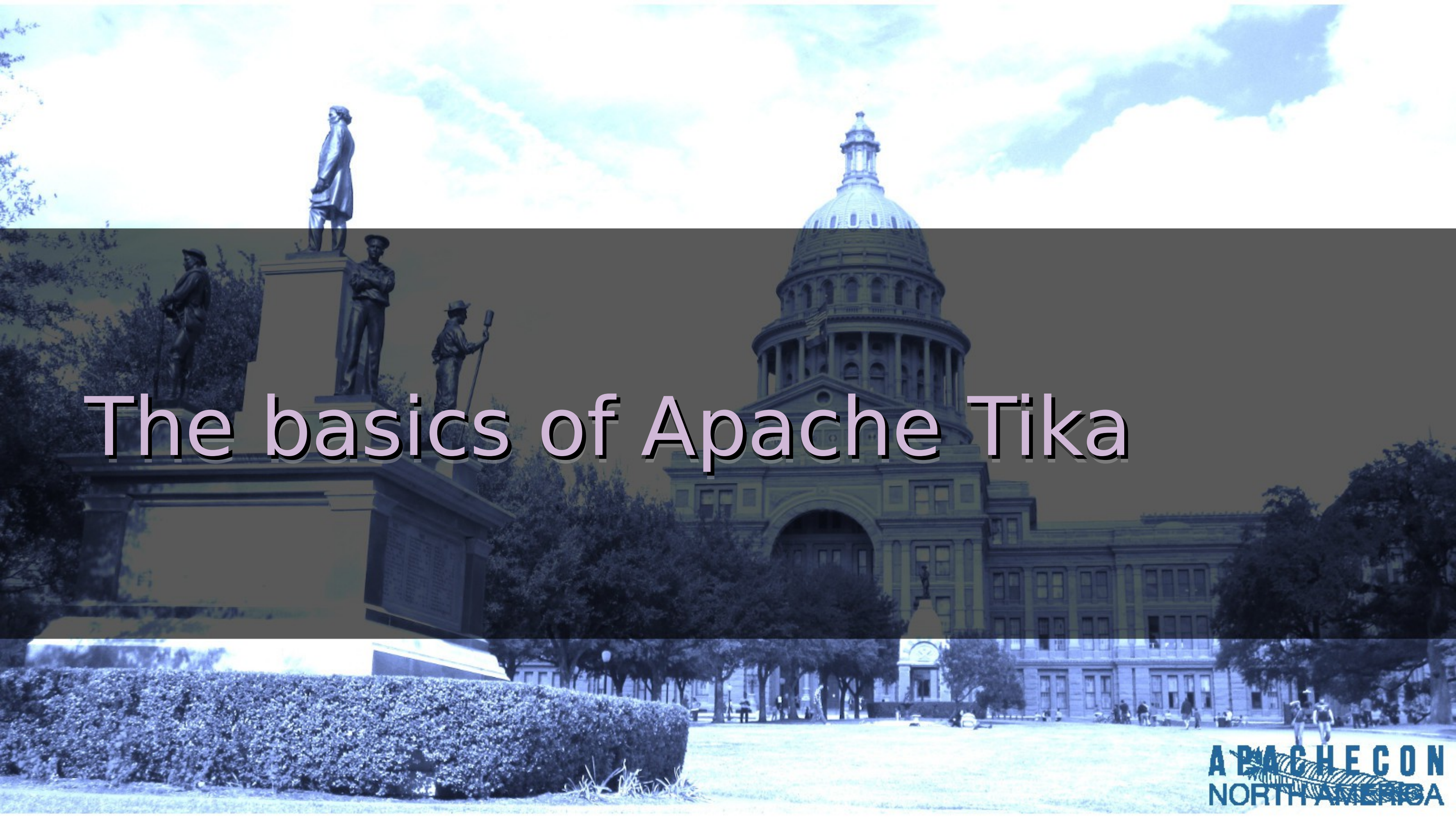
"small, yellow and leech-like, and probably the oddest thing in the Universe"

- Like a Babel Fish for content!
- Helps you work out what sort of thing your content (1s & 0s) is
- Helps you extract the metadata from it, in a consistent way
- Lets you get a plain text version of your content, eg for full text indexing
- Provides a rich (XHTML) version too



(Some) Supported Formats

- HTML, XHTML, XML
- Microsoft Office – Word, Excel, PowerPoint, Works, Publisher, Visio – Binary and OOXML formats
- OpenDocument (OpenOffice)
- iWorks – Keynote, Pages, Numbers
- PDF, RTF, Plain Text, CHM Help
- Compression / Archive – Zip, Tar, Ar, 7z, bz2, gz etc
- Atom, RSS, ePub   Lots of Scientific formats
- Audio – MP3, MP4, Vorbis, Opus, Speex, MIDI, Wav
- Image – JPEG, TIFF, PNG, BMP, GIF, ICO



The basics of Apache Tika

Detection

- Work out what kind of file something is
- Based on a mixture of things
  - Filename
  - Mime magic (first few hundred bytes)
  - Dedicated code (eg containers)
  - Some combination of all of these
- Can be used as a standalone – what is this thing?
- Can be combined with parsers – figure out what this is, then find a parser to work on it

Metadata

- Describes a file
- eg Title, Author, Creation Date, Location
- Tika provides a way to extract this (where present)
- However, each file format tends to have its own kind of metadata, which can vary a lot
- eg Author, Creator, Created By, First Author, Creator[0]
- Tika tries to map file format specific metadata onto common, consistent metadata keys
- “Give me the thing that closest represents what Dublin Core defines as Creator”

Plain Text

APACHECON NORTH AMERICA

- Most file formats include at least some text
- For a plain text file, that's everything in it!
- For others, it's only part
- Lots of libraries out there which can extract text, but how you call them varies a lot
- Tika wraps all that up for you, and gives consistentency
-
- Plain Text is ideal for things like Full Text Indexing, eg to feed into SOLR, Lucene or ElasticSearch

XHTML

APACHECON NORTH AMERICA

- Structured Text extraction
- Outputs SAX events for the tags and text of a file
- This is actually the Tika default, Plain Text is implemented by only catching the Text parts of the SAX output
- Isn't supposed to be the “exact representation”
- Aims to give meaningful, semantic but simple output
- Can be used for basic previews
- Can be used to filter, eg ignore header + footer then give remainder as plain text

Detecting File Types

Isn't that simple?

- Surely you just know what a file is on your computer?
- Well, probably on your computer, and maybe elsewhere?
- OK, so maybe people rename things, but it's close, no?
- Ah, the internet... But that's normally right isn't it?
- Hmm, well most web servers tell the truth, right?
- They wouldn't get it that wrong?
- A few percent of the internet, that's hardly that much?
- And people would never rename things by accident?
- Operating Systems would never “help”, would they?

Filenames

- Filenames – normally, but not always have extensions
- There aren't that many extension combinations
- There are probably more file formats than that
- No official way to reserve an extension
- So everyone just picks a “sensible” one, and hopes that don't have (too many) clashes...
- What happens if you rename a file though?
- Or have a file without one?
-
- Quick, but dirty, and may not be right...

Mime Magic

- Most file formats have a well known structure
- Most of these have a (mostly) unique pattern near the start of them
- These are often called Mime Magic Numbers
- In some cases, these are numbers
- More commonly, they're some sort of number or text or bit mask
- Ideally located at a fixed offset, even better, right at the start of the file
- But not always...

More on Magic

- PDFs (should) start with %PDF-
- Microsoft Office OLE2 docs start with 0xd0cf11e0a1b11ae1
- Most Zip files start with PK\003\004
- AIFF starts with FORM????AI(FF|FC) (mask 5-8)
- PE Executables normally have PE\000\000 at 128 or 240

- Not all of these are true constants
- Not all of these are unique - 0xfffe can be UTF-16LE or MP3
- Container formats – Zip can be Zip, OOXML, iWorks etc

Containers

- Some file formats are actually containers, and can hold lots of different things in them
- For example, a .zip file could just be a zip of random files
- Or it could be a Microsoft OOXML file (eg .docx, .pptx)
- Or it could be an OpenDocument Format file (eg .ods)
- Or it could be an iWorks file (Numbers, Pages, Keynote)
- Or it could be an ePub file
- Or....
- A .ogg could be audio, video, text, or many!

Dealing with Containers

- We can use Mime Magic to detect the container itself
- But to go beyond that, we need to actually parse the container, and look for special entries
- All the container based formats have some way of identifying the contents, of varying difficulty
-
- eg Check zip entries, look for _rels/.rels, it's OOXML, then read that file to get the mimetype
- eg Check Ogg for CMML, use that to get the primary stream type, else count stream types

Pull it all together

- If you want, you can call all of these elements individually
- For example, you might want low quality but fast detection only, so you call only the filename matching
-
- For most things, you want the best quality detection that Tika is able to deliver
- Tika uses a Service Loader mechanism to find available detectors, and then weighs up their outputs
- Ask TikaConfig for this, or load your own manually
- DefaultDetector tries them all for you

Detecting Text

Encodings 101

- Many different ways to encode text in a file
- “A” could be 0x41 (ascii,utf8 etc), 0x0041 (utf16le), 0x4100 (utf16be), 0xC1 (ebcdic)
- 1-byte-per-character encodings historically very common
- But means that you need lots of different encodings to cope with different langauges and character sets
- 0xE1 – could be: á α с ב ف แ (or something else too!)
- Some formats include what encoding they've used
- But many key ones, including plain text, do not!

Languages

- Different languages have different common patterns of letters, based on words, spellings and patterns
- If you see accents like á è ç then it probably isn't English
- If you see a word starting with an S, it probably isn't Spanish, but if you see lots starting “ES” it might be
- You can look for these patterns, and use those to identify what language some text might be in
- Really needs quite a bit of text to work on though, it's very hard to make meaningful guesses on just a few letters!

n-grams

- Wikipedia says “An n-gram model is a type of probabilistic language model for predicting the next item in such a sequence in the form of a (n - 1)–order Markov model”
- Basically, for us, it's all the possible character combinations (including start + end markers) along with their frequency
- The “n” is the size
- Trigrams of “hello” are [ ]he, ell, llo, lo[ ]
- Quadgrams of “hello” are [ ]hel, ello, llo[ ]
- Can be used to identify both language and encoding

Detecting with Tika

APACHECON NORTH AMERICA

- Tika will detect the encoding of text automatically when parsing
- Encoding will be placed into the Metadata object
- Text will be returned as Java Strings (unicode)
- Encoding can be explicitly detected with EncodingDetector
- Several Encoding Detection methods supported, ICU4J main
- Most major encodings and languages are supported
- Language can be detected with LanguageIdentifier
- About 30 languages supported out of the box, more can be added in if you build ngram profiles

False Positives, Problems

- For encoding detection to work, Tika needs to recognise the file as Plain Text
- Too many control characters near the start can cause Tika to decide it isn't Plain Text, so won't detect
- Some encodings are very similar, hard to tell apart
- For short runs of text, very hard to be sure what it is
- Same pattern can crop up in different languages
- Same pattern could occur between different languages when in different encodings

Embedded Resources

Not just containers!

APACHECON NORTH AMERICA

- Container formats like Zip contain embedded resources, the files within them
- Many office documents support embedded resources too
- Microsoft Office documents can have other documents embedded within them, eg Excel in PowerPoint
- Most of the document formats support embedding images within the document, and many video too
- Where possible, Tika gives you the embedded resource, and indicates in the XHTML where it came from

Picking what to get

- Driven from the ParseContext

parse(InputStream,ContentHandler,Metadata,ParseContext)

- Parsers finding embedded resources fetch a EmbeddedDocumentExtractor from the ParseContext
- They supply as much information as is available, such as the Filename and the Content Type

shouldParseEmbedded(Metadata)

- Your code then decides if it wants this embedded resource, or if it wants to skip over it

Saving and recursing

- If you requested it, your EmbeddedDocumentExtractor will be called for the resource

  parseEmbedded(InputStream,ContentHandler,Metadata,bool)
- To save the resource, just grab the details about it from the Metadata object, and stream the InputStream
- To process the resource, just grab an AutoDetectParser (or similar), and pass it the resource
- To recurse, do the same, but pass along the same ParseContext you started with
- TikaCLI has a good example of all this

Extending Tika for new file formats

Custom Mimetypes

- Tika has a large number of built in definitions for different mimetypes, currently >1400
- Not all of them have mime magic, and not all possible public formats are defined, let alone private ones
- It's possible to define your own custom mimetypes, with or without magic, and have these loaded and used by Tika when it's doing detection
- Can have multiple custom mimetypes per file, and multiple files on the classpath (eg one per jar)
- org/apache/tika/mime/custom-mimetypes.xml

Custom Mimetypes

```
<?xml version="1.0" encoding="UTF-8"?>
<mime-info>
  <!-- A mimetype without any matching or globbing -->
  <mime-type type="hello/world" />

  <!-- A more complex mimetype, with a glob and a match -->
  <mime-type type="hello/world-file">
     <_comment>A "Hello World" file</_comment>
     <hello>world</hello>
     <glob pattern="*.hello.world" />
     <magic priority="50">
        <match value="Hello, World!" type="string" offset="0:13" />
     </magic>
     <sub-class-of type="hello/world" />
  </mime-type>
</mime-info>
```

Custom Parsers

APACHECON NORTH AMERICA

- Two key methods to implement

parse(InputStream,ContentHandler,Metadata,ParseContext)

getSupportedTypes(ParseContext)

- Parser should read the input, pass it to another library if needed, output XHTML SAX events for the textual and structural parts of the file, populate the Metadata, and offer recursion if it finds embedded resources
- Simple example available on the Tika website (13 line!)

http://tika.apache.org/1.5/parser_guide.html

When you need more control

What Tika tries to do

- Tika aims to map file format specific metadata onto a common, consistent set, based on well known standards
- Tika aims to provide semantically meaningful, but not cluttered XHTML
- It isn't supposed to be a perfect rendition of the original file into XHTML
- It is supposed to be simple and clean
- eg Word Parser reports tables and style names, but not Word-style HTML full of fonts +colours
- eg Excel Parser returns what's there, not a CSV

Overriding Parsers

- If there are two parsers which both claim to support the same mime type, DefaultParser will use the non-Tika one in preference, as long as it can be loaded
- Include a org.apache.tika.parser.Parser service file with your parser, then it can be auto-loaded
- You can explicitly initialise a TikaConfig or DefaultParser with a different Parsers list, to control what ones are and aren't available
- You can have a Tika Config XML file to control parser use
- If you want, you can call individual parsers directly

Overriding Parser Config

```
<properties>
  <parsers>
    <parser class="org.apache.tika.parser.DefaultParser">
      <mime-exclude>image/jpeg</mime-exclude>
      <mime-exclude>application/pdf</mime-exclude>
      <parser-exclude class="org.apache.tika.parser.executable.ExecutableParser"/>
    </parser>
    <parser class="org.apache.tika.parser.EmptyParser">
      <mime>application/pdf</mime>
    </parser>
  </parsers>
</properties>
```

Doing your own thing

- Look at the source code of the Tika parser, and see if you can override it to do something different in places
- Request enhancements if you think there's something that ought to be easier to override, or ought to be an option
- Not everything will be accepted though...
- Tika only uses permissive licensed libraries, may be commerical or copyleft ones too you want to use

wiki.apache.org/tika/3rd%20party%20parser%20plugins

- Look at examples of the upstream libraries
- Write your own parser, register, use!

Ways to run Tika

Overview of Options

- Tika-App – command line tool
- Pure Java – Tika Facade – simple way from Java
- Pure Java – Direct – full control over what happens
- OSGi – all your dependencies nicely wrapped up
- Forked Parser – parsing in a different JVM
- JAX-RS Network Server – RESTful interface to Tika
- SOLR Plugin – Tika parsing from within SOLR upload
- Anything you want to write yourself!
- (These are just the main ways)

Tika-App

APACHECON NORTH AMERICA

- Single runnable jar, containing all of Tika + dependencies
- GUI mode, ideal for quick testing and demos
- CLI mode, suitable for calling from non-Java programs
  - Detection: --detect
  - Metadata: --metadata
  - Plain Text: --text
  - XHTML: --xml
- Information on available mimetypes, parsers etc
- JVM startup costs means not the fastest for lots of docs

Pure Java – Tika Facade

- Very simple way to call Tika
- Tika facade class – org.apache.tika.Tika
- Detection from streams, bytes, files, urls etc
- Parsing to Strings, Readers, Plain Text or XHTML
- By default, uses all available parsers and detectors

- Requires that all the Tika jars, and their dependencies are present on the classpath, and don't clash
- Harder to check if that hasn't worked properly, eg missing jars or jar clashes

Pure Java - Direct

- Slightly more lines of code, but you get full control
- Normally start with org.apache.tika.TikaConfig
- From that fetch Detectors, Parsers, Mime Types list etc
- Typically want to use AutoDetectParser to actually parse
- If possible, wrap your content with a TikaInputStream before parsing – create from InputStream or File

- Still needs all jars on classpath
- Ask DefaultParser what Parsers + mimetypes are available

OSGi

- Tika ships an optional OSGi bundle
- You need both tika-core and tika-bundle on your classpath
- Start the Tika bundle and Tika Core, then Tika will be able to detect the available parsers + detectors and use
- Can then either use the Tika facade, or classes directly

- Avoids issues with forgetting jars, jar clashes etc
- But needs an OSGi environment setting up first
- And not all that widely used (but it does work, see the unit tests for an example!)

Forked Parser

- By default, Tika runs in the same JVM as the rest of your code, so if Tika (or a library it depends on) breaks, so does the rest of your JVM
- This shouldn't happen, but some broken files can trigger parsers to run out of memory, or loop forever
- If you're doing internet scale things, this becomes all the more likely to be hit eventually!
- Forked Parser fires up a new JVM, sends over Tika + dependencies, then communicates with that JVM to do the parsing
- If this crashed, doesn't affect your main JVM

JAX-RS Network Server

- JAX-RS – JSR-311 RESTful network server
- Powered by Apache CXF
- The tika-server jar can be run, starts up the server
- Content is sent via HTTP PUT calls to service URLs
- Supports detection, metadata extraction, plain text, xhtml, and a few other things too
- Somewhat self documenting, that's set to improve soonish
- See https://wiki.apache.org/tika/TikaJAXRS
- Recommended way to call Tika from non-Java languages
- Runs in a new JVM

SOLR Plugin

- ExtractingRequestHandler – SOLR plugin to call Tika
- Send your content to /solr/update/extract and it'll be passed to Tika before being indexed
- Use SOLR config to control what goes where, eg which bits of metadata to index

- Maintained by the SOLR community, not the Tika one
- Requires all the Tika jars + dependencies on classpath, very limited ways to check if that's all there or not
- Not all Tika functionality is exposed (eg recursion)

Big Data it!
Tika at Scale

Lots of Data is Junk

- At scale, you're going to hit lots of edge cases
- At scale, you're going to come across lots of junk or corrupted documents
- 1% of a lot is still a lot...
- Bound to find files which are unusual or corrupted enough to be mis-identified
- You need to plan for failures!

Unusual Types

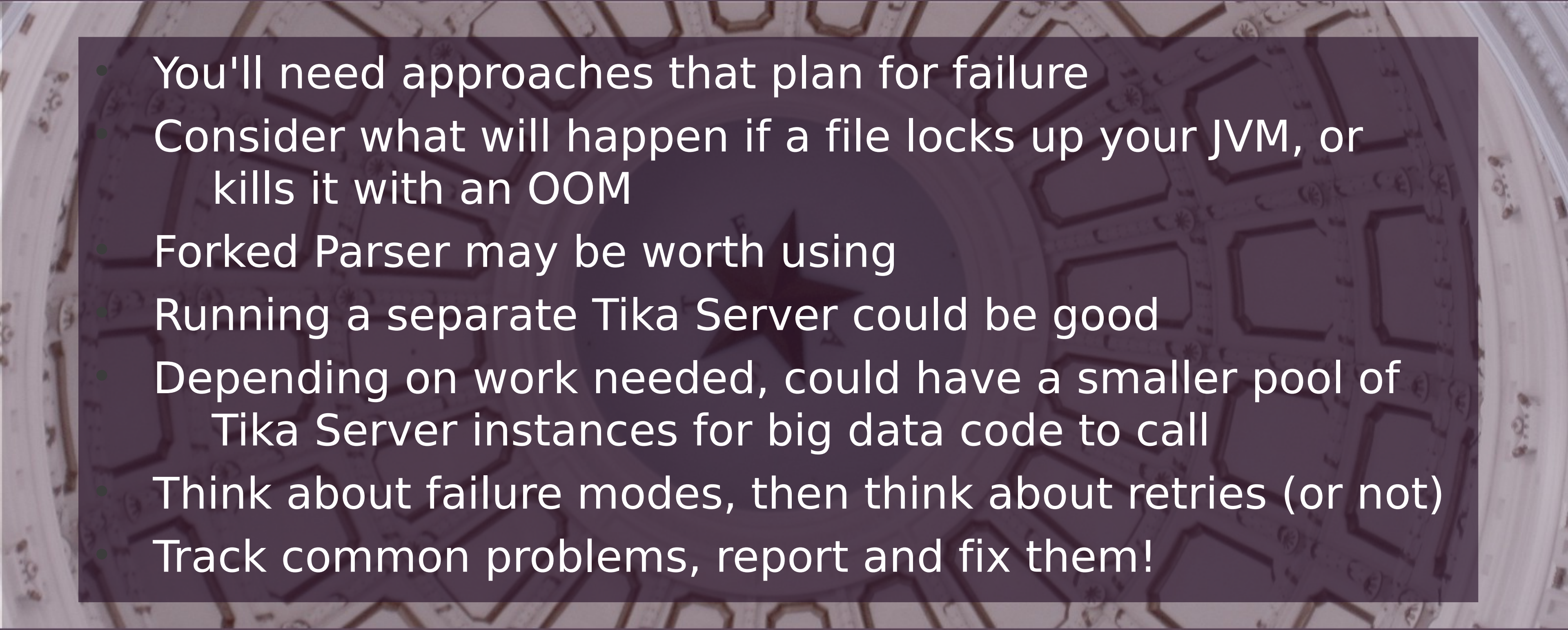
- If you're working on a big data scale, you're bound to come across lots of valid but unusual + unknown files
- You're never going to be able to add support for all of them!
- May be worth adding support for the more common “uncommon” unsupported types
- Which means you'll need to track something about the files you couldn't understand
- If Tika knows the mimetype but has no parser, just log the mimetype
- If mimetype unknown, maybe log first few bytes

Failure at scale

- Tika will sometimes mis-identify something, so sometimes the wrong parser will run and object
- Some files will cause parsers or their underlying libraries to do something silly, such as use lots of memory or get into loops with lots to do
- Some files will cause parsers or their underlying libraries to OOM, or infinite loop, or something else bad
- If a file fails once, will probably fail again, so blindly just re-running that task again won't help

Failure at scale continued

APACHECON NORTH AMERICA



- You'll need approaches that plan for failure
- Consider what will happen if a file locks up your JVM, or kills it with an OOM
- Forked Parser may be worth using
- Running a separate Tika Server could be good
- Depending on work needed, could have a smaller pool of Tika Server instances for big data code to call
- Think about failure modes, then think about retries (or not)
- Track common problems, report and fix them!

Bringing it Together: Tika Batch, Tika Batch Hadoop and Tika Eval

APACHECON NORTH AMERICA

Tika Batch - TIKA-1330

- Aiming to provide a robust Tika wrapper, that handles OOMs, permanent hangs, out of file handles etc
- Should be able to use Tika Batch to run Tika against a wide range of documents, getting either content or an error

- First focus is on the Tika App, disk-to-disk wrapper still baking at https://github.com/tballison/tika/tree/TIKA-1302
- Next step is for the Tika Server, to have it log errors, provide a watchdog to restart after serious errors etc
- Accept there will always be errors! Work with that

Tika Batch Hadoop

- Once we have the basic Tika batch working...
- Aiming to provide a Hadoop implementation as well
- Will process a large collection of files, providing either Metadata+Content, or a detailed error of failure
- Failure could be machine/enivornment, so probably need to retry a failure incase it isn't a Tika issue!
- Will be partly inspired by the work Apache Nutch does
- Tika will “eat our own dogfood” with this, using it to test for regressions / improvements between versions

Tika Eval - TIKA-1332

- Building on top of Tika Batch, to work out how well / badly a version of Tika does on a large collection of documents
- Provide comparible profiling of a run on a corpus
- Number of different file types found, number of exceptions, exceptions by type and file type, attachements etc
- Also provide information on langauge stats, and junk text
- Identify file types to look at supporting
- Identify file types / exceptions which have regressed
- Identify exceptions / problems to try to fix
- Identify things for manual review, eg TIKA-1442 PDFBox bug

Batch+Eval+Public Datasets

- When looking at a new feature, or looking to upgrade a dependency, we want to know if we have broken anything
- Unit tests provide a good first-pass, but only so many files
- Running against a very large dataset and comparing before/after the best way to handle it

- Initially piloting + developing against the Govdocs1 corpus http://digitalcorpora.org/corpora/govdocs
- Using donated hosting from Rackspace for trying this
- Need newer + more varied corpuses as well! Know of any?

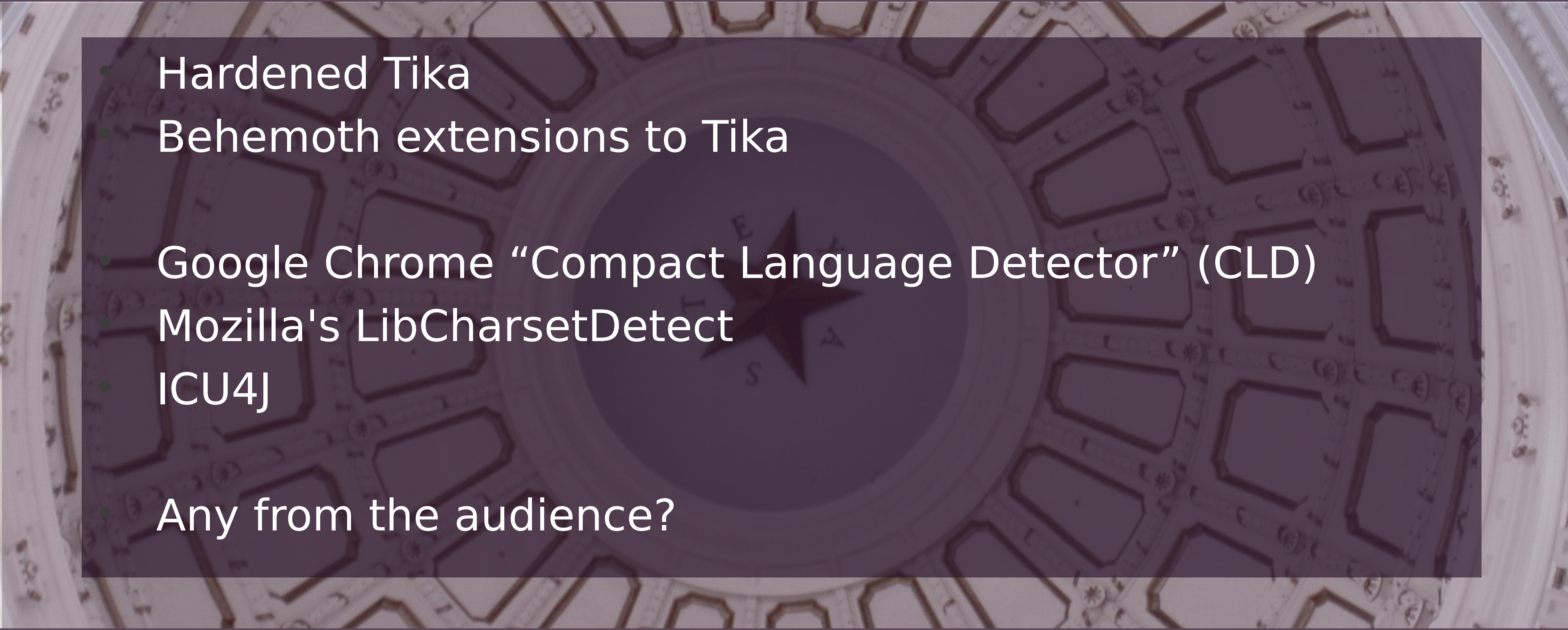
Friends of Tika
Other projects to look at

Other Apache Projects

APACHECON NORTH AMERICA

- Nutch
- Any23
- OpenNLP
- UIMA
- CTAKES
- Any from the audience?

External Projects



- Hardened Tika
- Behemoth extensions to Tika

- Google Chrome “Compact Language Detector” (CLD)
- Mozilla's LibCharsetDetect
- ICU4J

- Any from the audience?

Any Questions?

APACHECON NORTH AMERICA

Nick Burch

@Gagravarr
nick@apache.org

APACHECON
NORTH AMERICA