SALSA: A success story of migrating research ideas into industry

Michael W. Godfrey
SWAG, University of Waterloo
migod@uwaterloo.ca

J2EE architecture analysis using relational algebra

Michael W. Godfrey
SWAG, University of Waterloo
migod@uwaterloo.ca

Lossy program analysis or Lies my extractor told me

Michael W. Godfrey
SWAG, University of Waterloo
migod@uwaterloo.ca

Research pre-history

- 2002 a research project called RAMP is born
  - RAMP == Rapid Assisted Migration Project
  - An industrial research collaboration with Sun Microsystems
  - Principal investigators:
    - UW: Profs Holt / Malton / Godfrey
    - Sun: Brian Down, Wai-Ming Wong
  - Part of the CSER research consortium:

    `http://www.cser.ca`

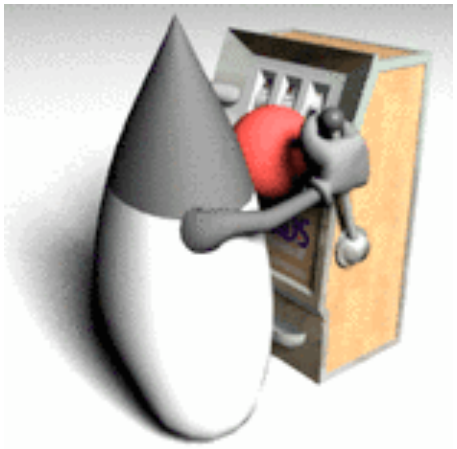
RAMP to Jackpot

- Contacts thru Sun / RAMP / conferences led to a sabbatical invitation
  - I spent Sept-03 to Aug-04 in Sun Research Labs in Mountain View, CA. It was awesome.

- I went to work on Jackpot, an AST-based analysis tool
  - Team members:
    - Michael Van De Vanter, James Gosling, Tom Ball, Tim Prinzing

Sun's Jackpot Tool

- AST-based analysis + transformation tool
  - Metrics summaries
  - "Bad smell" detection
  - Semi-automated source transformation
    - J++-to-Java migration, bad smell removal, …
  - Code visualization
  - "Smart" editor support



- Basic idea:
  1. Suck up *whole program* into memory
  2. "Play" with the AST
  3. Output transformed source code

Hello, Jackpot

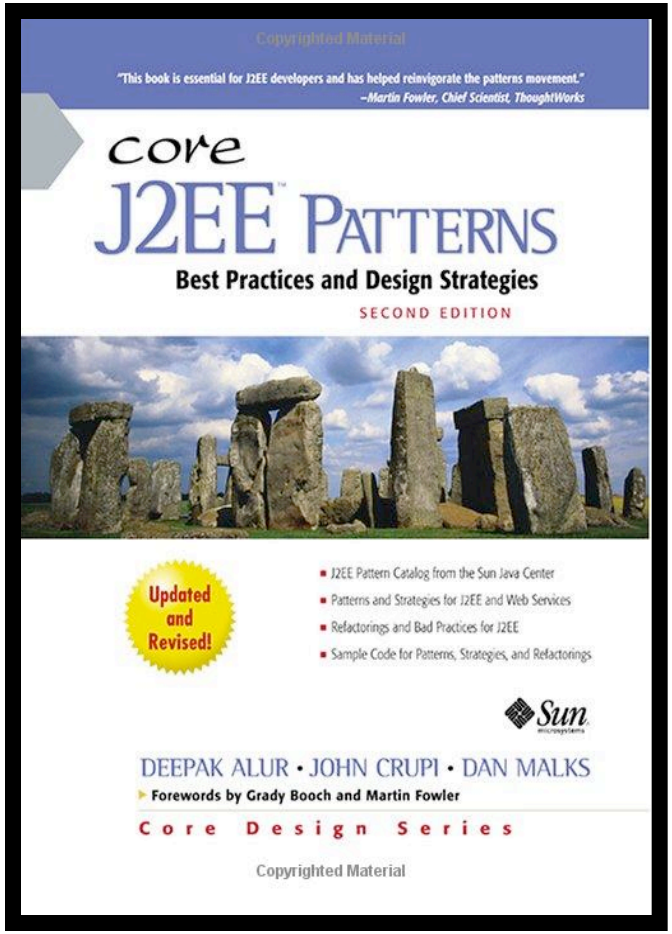
- When I arrived in Sept 2003:
  - Basic infrastructure works
  - Several bad smells can be detected automatically
  - Several automated transformations work
  - ...
- But
  - While the technology is very promising, it's hard for outsiders to pick up and adapt easily
    - Must understand both Jackpot and `javac` internals
    - Work is slow going and very detailed (AST hacking)

And then …

- Corporate re-org in early fall 2003
  - Oh no!
  - James Gosling becomes CTO of tools and the team moves with him, effectively putting Jackpot on back burner for now
  - I'm all alone in the lab with 10 months to go in my sabbatical …
- I continued to play around with Jackpot, writing a few small plugins for it.
  - For fun, I started to implement a Java "fact extractor" based around the **`javac`** AST

And then …

- Van De Vanter introduces me to Sun DE John Crupi, who has a problem:
  - *"We wrote the book on J2EE patterns (good and bad), but we're still using* ***`grep`*** *and* ***`perl`*** *to fix them !"*
- Van De Vanter, Crupi, and I meet several times to sketch out the design of a prototype J2EE architecture analysis tool based around Jackpot
- I was charged with finishing up the fact extractor and writing a few "smell detection" scripts



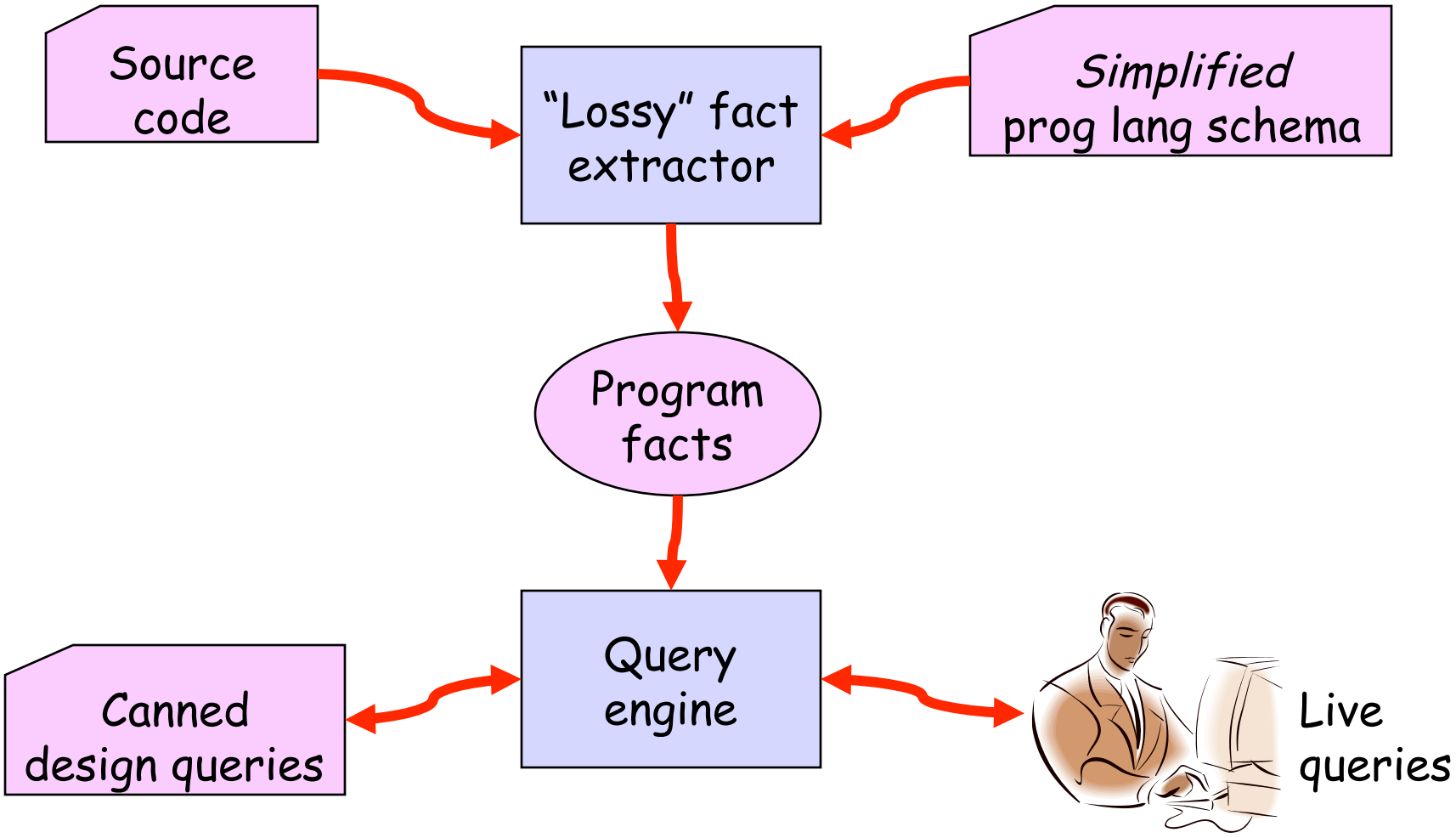
Kinds of program analysis tools

1. Special purpose, batch static analysis tools
   - Read in code, analyze, spit out (small) result set
     - Result set typically makes no sense on its own; need refs back to source code
   - Analysis goals hard-coded into tool
     - New goals? Write a new tool!

Kinds of program analysis tools

2. [Whole earth / big bang] [analysis / transform] tools:
   - Perform generic analysis (*e.g.,* compilation) and keep all of compilation "facts" in store
     - Then allow AST walkers to generate desired info
     - Source-to-source code transformation also possible
   - Analysis results can be customized via new tree walkers
     - Slow and detailed work
     - … but you can do just about anything to the source code
   - Each run requires a new compilation (or reading in saved AST / symbol table)

Lossy program analysis



Kinds of program analysis tools

3. "Lossy" program analysis
   - Generates a set of "facts" about the program
     - An abstracted ("lossy") view of the system, according to a defined schema
     - The facts are complete, relative to their defined abstraction level

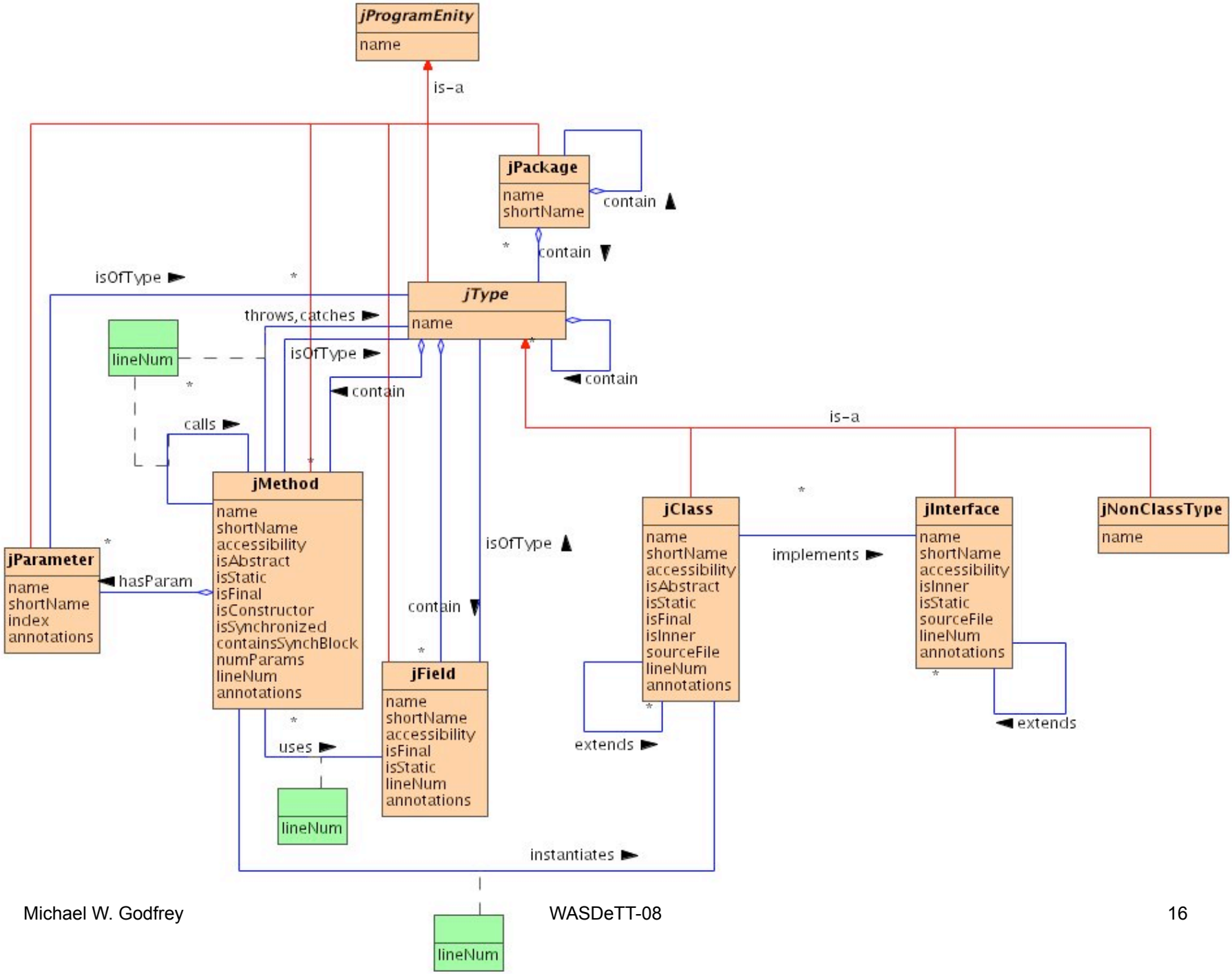
       e.g., can spot global variable uses across packages, but no information about how for loops are used
     - Source code examined only once
       - New run of the tool means only loading the "facts" into the query engine
       - Can add / refine queries using same factbase (since the facts don't change unless the code does)

“Lossy” program analysis

- Advantages:
  - Much easier to write canned queries, GUIs for navigation, experiment / go fishing with results
  - Model is self-contained, complete so no need to consult or link back to source code
  - Source code examined only once!
    - Loading factbase usually much faster than compilation

- Disadvantage:
  - Source-to-source code transformation not possible
    - But can feed results back into a whole-earth analsyis tool

    e.g., find known bad smells, feed fixes to a transformation engine


jProgramEnity
name
is-a
jPackage
name
shortName
contain
contain
isOfType
jType
name
throws,catches
lineNum
isOfType
contain
contain
calls
is-a
jMethod
name
shortName
accessibility
isAbstract
isStatic
isFinal
isConstructor
isSynchronized
containsSynchBlock
numParams
lineNum
annotations
jParameter
name
shortName
index
annotations
hasParam
isOfType
contain
jField
name
shortName
accessibility
isFinal
isStatic
lineNum
annotations
uses
lineNum
jClass
name
shortName
accessibility
isAbstract
isStatic
isFinal
isInner
sourceFile
lineNum
annotations
implements
extends
jInterface
name
shortName
accessibility
isInner
isStatic
sourceFile
lineNum
annotations
extends
jNonClassType
name
instantiates
lineNum

Jackpot-to-SALSA

- I “finish” my extractor (still part of Jackpot), and give a demo for Crupi’s group
  - I show how to define and run pattern queries they specify (using grok/QL) on source code they’ve provided

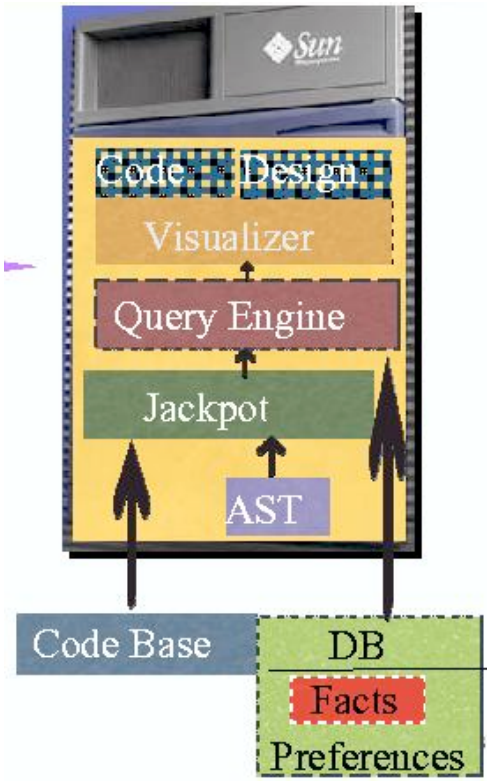
```
// Want to find all SessionBeans that call EntityBeans

extendsRTC = extends*
subtypeof = extendsRTC + extendsRTC o implements o extendsRTC
sessionsBeanClasses = classes ^
                      subtypeof . {"javax.ejb.SessionBean"}
entityBeanClasses = classes ^ subtypeof . {"javax.ejb.EntityBean"}
sessionBeansCallingEntityBeans = sessionsBeanClasses o calls
                                 o entityBeanClasses
```

SALSA goals *[Crupi]*

- Crupi pitches the idea to several big clients
  - It is very enthusiastically received!
  - The SALSA project (*Sun Appliance for Live Software Analysis*) is born!

- Main goal:
  - (Semi-) automate architectural assessment as much as possible
    - Aim for remote, collaborative, client-driven, early feedback
    - Build a library of known “J2EE bad patterns” + allow application/domain knowledge to be added
    - Feedback into the code (comments, annotations, transformed source code)



Later on …

- I finished the fact extractor
  - Now, a standalone Java 1.5 application
    - If `javac` can compile your code, I can extract it!
  - Extracts info about generic classes/methods, inner (non-local) classes, exceptions, initialization clauses, parameters, …
- Later work at UWaterloo
  - A co-op student who worked on Jackpot later completed a byte-code extractor using same schema
- Work on SALSA continued at Sun for a while after
  - Patterns library
  - GUI
  - Infrastructure enhancements
  - … then most of the the SALSA team left to join a startup ☹
  - … and later still Jackpot was resurrected and integrated into Netbeans

Things I learned

- Working code is better than an unproven but elegant idea
  - "Do the simplest thing that could possibly work!"
  - Don't over-engineer; get it working and get feedback
    - You probably got it wrong anyway
    - I spent weeks working on modelling inner classes …
- Don't expect to be able to compete with their development skills
  - You're a researcher, not a professional developer. There's a difference!
  - Ask for help when you need it … it will save a lot of time and build trust in your honesty

Things I learned

- Standards tools beat the pants off of elegant research prototypes
  e.g., grok vs. SQL … guess which one the developers were more comfortable with!

- Office politics is real
  - Sometimes, need to work in stealth mode
  - Good ideas get spiked for bad reasons … deal with it and move on
  - Good ideas get spiked for good reasons too

Things I learned

- Solving a real problem is what industry wants!
  - Any given research idea “might work”, but what are your clients actually interested in? What can’t they do well that you can?
  - Ask, then do a lot of listening.
  - Don’t expect new research to come from this … but keep in mind that the experience alone is probably a pretty good paper.

- Work towards them, don’t expect them to work towards you
  - They probably don’t understand what research is, and they don’t (and shouldn’t) really care
  - Don’t expect to meet them half way. You’re on their turf! When in Rome …

Things I learned

- You need a high-level champion
  - … who believes in the project, and will fight for it
    e.g., resources, personnel, access to infrastructure
  - … even if he/she doesn't really understand it

- You need a high-level technical interactor
  - … who can define goals, evaluate progress, suggest contacts, provide access
  - Or else: "Hey, it's your turn to babysit the researcher!"
  - Regular meetings are a must
    - I worked in a strange solo-world most of the time

Things I learned

- They need to understand that they will have to take ownership eventually
  - If it goes well, that is.
  - You should be so lucky!

SALSA: A success story of migrating research ideas into industry

Michael W. Godfrey
SWAG, University of Waterloo
migod@uwaterloo.ca