Velocity in Research

CS 197 | Stanford University | Michael Bernstein

What problem are we solving?

"Research is so much slower than industry."

"I feel like we're just not getting anywhere."

"This keeps dragging on and it's not working. I'm losing motivation."

"I missed another submission deadline. I think my advisor is starting to lose faith."

2

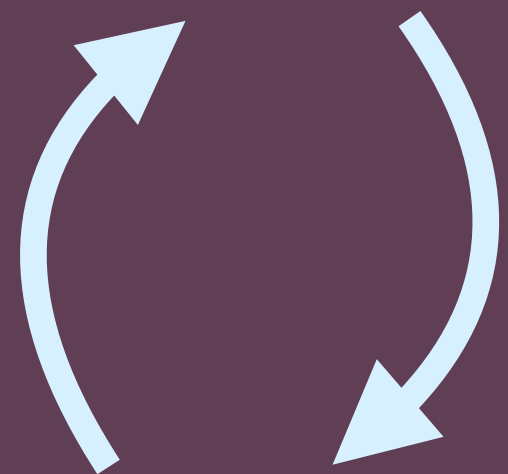# Today's big idea: velocity

What is research velocity?

How do we achieve high velocity?

What other signals do people mistake for velocity?

Bernstein theory of faculty success

To be a Stanford-tier faculty member, you need to master two skills that operate in a tight loop with one another.

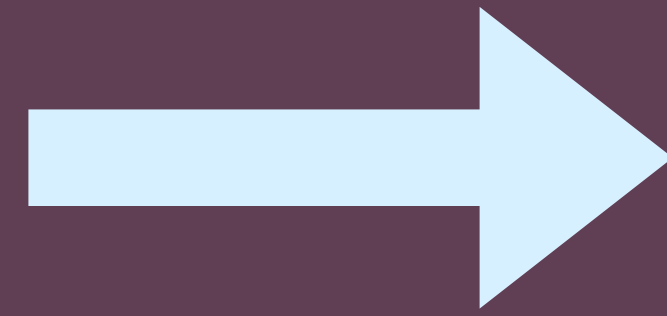**Vectoring**: identifying the biggest dimension of risk in your project right now

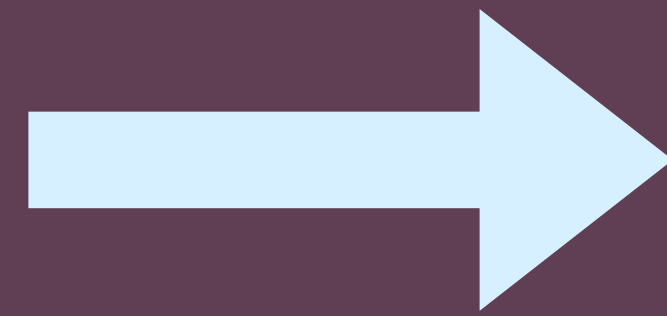**Velocity**: rapid reduction of risk in the chosen dimension

What Is Velocity?

Problematic point of view
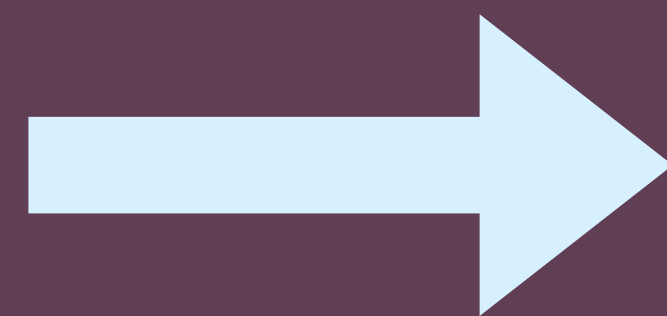
"Research is so much slower than industry." ➡️

"I feel like we're just not getting anywhere." ➡️ **We're not making enough progress.**

"I missed another submission deadline." ➡️

What research is not

1. Figure out what to do.

2. Do it.

3. Publish.

What research is

Research is an iterative process of exploration, not a linear path from idea to result [Gowers 2000]

My diagnosis: The Swamp

I have led and advised many projects at this point, and I can now say with certainty: nearly every project has a swamp.

The Swamp: challenges that get the project stuck for an extended length of time

Model not performing well

Design not having intended effect

Engineering challenges keep cropping up

&etc

Photo by Big Cypress National Preserve

Swamps make progress a poor measure

Swamps can make a project appear to have no or little progress for an extended period of time.

However, swamps are when you need to be at your most creative. You need to try many different ideas, and rapidly, to orienteer your way out of a swamp.

The difference between an amazing and a merely good researcher: how effectively and rapidly you explore ways to escape the swamp.

9

Enter velocity

Drawn from theory and practice of rapid prototyping

Buxton, Sketching User Experiences

Schön, The Reflective Practitioner

Houde and Hill, What Do Prototypes Prototype?

CS 247 (cs247.stanford.edu) — I realized that none of my PhD students have taken or TA'ed this class

"Enlightened trial and error succeeds over the planning of the lone genius." - Tom Kelley

Velocity vs. progress

Progress is an absolute delta of your position from the last time we met. How far have you gotten?

Velocity is a measure of the distance traveled in that time.

If you tried a ton of creative different ideas and they all failed…

that's low progress
but high velocity

I will be thrilled

Why is velocity a better measure?

Because we have likely learned a ton from the failures along the way.

Because we likely needed to experience those failures to eventually get to a success: you're learning the landscape.

Because the worst outcome is not failure, but tunneling unproductively.

That's low progress and low velocity

this is when I

How do I achieve high velocity?

Restating our goal, precisely

Each week's effort — a draft paper introduction, a user interface, an engineered feature, an evaluation design — is on the path toward understanding the research question.

We have a question to answer this week: Will our hunch work in a simple case? Is assumption X valid? Will this revised model overcome the problematic issue? Can we write a proof for the simple case? We've chosen this week's question that we're trying to answer carefully.

Velocity is the process of answering that question as rapidly as possible.

Choosing this question is the process of vectoring.

14

Approach: core vs. periphery

Achieving high velocity means sprinting to answer this week's question, while minimizing all other desiderata for now.

This means being clear with yourself on what you can ignore:

Core: the goal that needs to be achieved in order to answer the question

Periphery: the goals that can be faked, or assumed, or subsetted, or mocked in, so we can focus on the core.

Core-periphery mindset

The week's goal is **not a demo**.

Though this is what is tempting: think, select, and then create.

But this means working on everything both in the core and in the periphery.

The week's goal is instead an **answer to a question**.

To answer a question, you don't need to address all the issues in the periphery. Just focus on what's in the core.

Make strong assumptions about everything that's in the periphery: use an easy or smaller subset of the data, make simplifying assumptions while working on your proof, ignore other nagging questions for the moment

Core-periphery mindset

I'm dedicating a second slide to this concept because it's the key.

Your approach should be, necessarily, incomplete. Do not create a mockup or a scale model. Instead, derive everything from your current question:

Will this approach retain all users?
Will this measure correlate with my gut observations?
Will this engineering approach be satisfactory?

Be rapid. Be ruthless. Strip out or fake everything not required to answer the question.

Core-periphery mindset

Seriously: I'm dedicating a third slide to this.

Answer questions, don't engineer. This tends to rankle essentially every facet of your undergraduate training.

Too often, people pursue perfection in the first pass: perfect drafts, perfectly engineered software, perfect interaction design.

Remember: the goal is to answer the question, not to build that part of your system permanently (yet).

11:08 AM 09/03/96   4/50
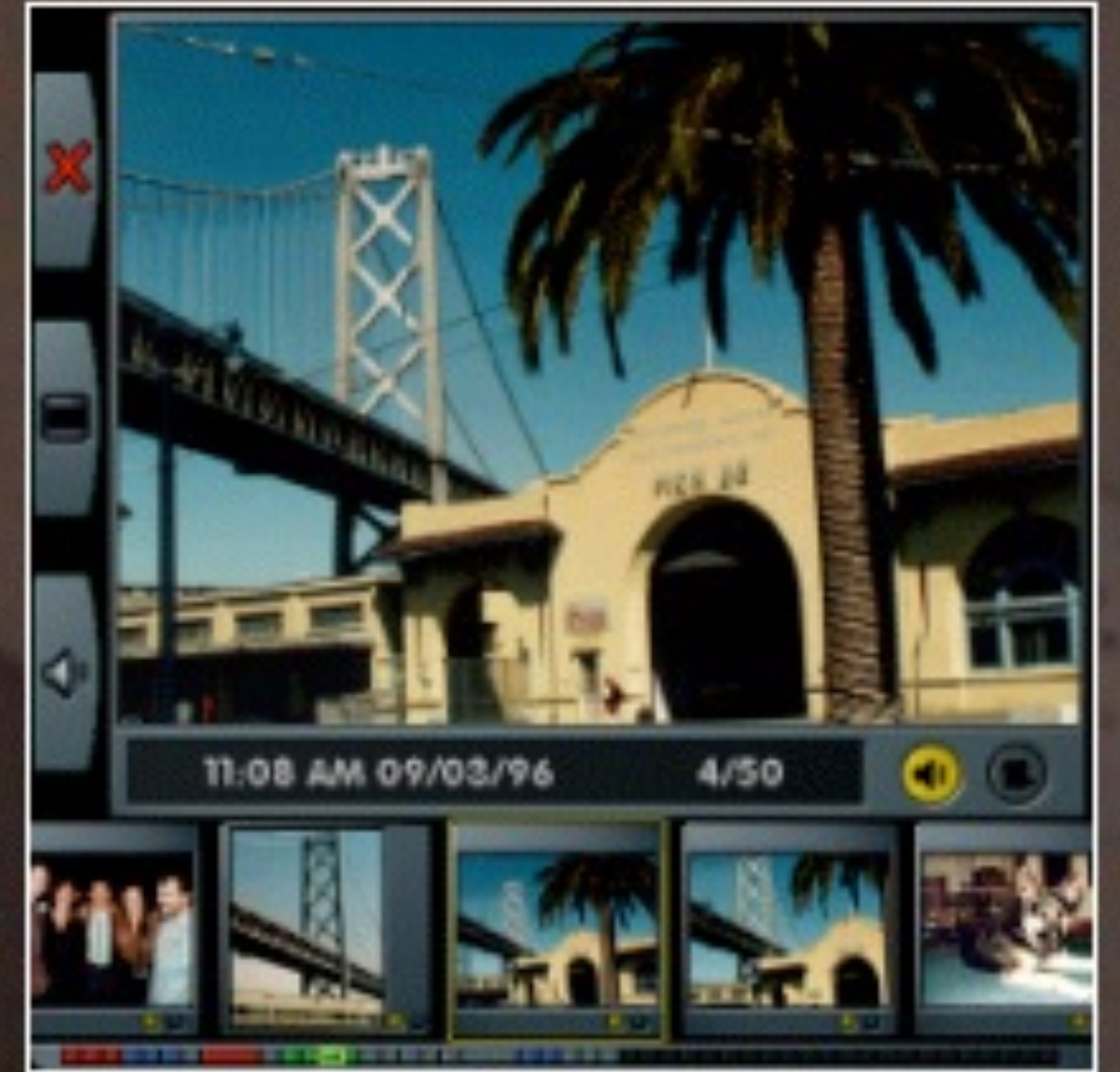
What question
were they asking?

What did they
trade off?

All together now

Each week, we engage in vectoring to identify the biggest unanswered question. This should be the focus of your velocity sprint for the week.

To hit high velocity, be strategic about stripping out all other dependencies, faking what you need to, etc., in order to answer the question.

Be prepared to iterate multiple times within the week!

Let's Try It

We'll try out…

A social debugging question

A design question

An engineering question

Get in groups of 3–4, you'll have two minutes to discuss each question.

Social debugging: flash organizations

We had a problem of online workers not being as good as their Upwork profile suggested. We wanted workers who were experts at Angular, Django, UI, UX, marketing, etc, but often in practice they were not as good as they advertised.

We had a hunch that giving workers ~1hr starter tasks would allow us to vet them.

How do we test this hunch?

Flash Organizations: Crowdsourcing Complex Work By Structuring Crowds As Organizations

Melissa A. Valentine, Daniela Retelny, Alexandra To, Negar Rahmati, Tulsee Doshi, Michael S. Bernstein
Stanford University
flashorgs@cs.stanford.edu

Social debugging: flash organizations

We picked a small number of domains and manually generated quick test tasks for them. We posted these as jobs, giving a time limit. We manually evaluated the results.

We didn't care about generalizability or software integration.

Afterwards, we asked ourselves: could this scale to hundreds of people and tens of domains?

Flash Organizations: Crowdsourcing Complex Work By Structuring Crowds As Organizations

Melissa A. Valentine, Daniela Retelny,
Alexandra To, Negar Rahmati, Tulsee Doshi, Michael S. Bernstein
Stanford University
flashorgs@cs.stanford.edu

ABSTRACT
This paper introduces *flash organizations*: crowds structured like organizations to achieve complex and open-ended goals. Microtask workflows, the dominant crowdsourcing structures today, only enable goals that are so simple and modular that their path can be entirely pre-defined. We present a system that organizes crowd workers into computationally-represented structures inspired by those used in organizations — roles, teams, and hierarchies — which support emergent and adaptive coordination toward open-ended goals. Our system introduces two technical contributions: 1) encoding the crowd's division of labor into de-individualized roles, much as movie crews or disaster response teams use roles to support coordination between on-demand workers who have not worked together before; and 2) reconfiguring these structures through a model inspired by version control, enabling continuous adaptation of the work and the division of labor. We report a deployment in which flash organizations successfully carried out open-ended and complex goals previously out of reach for crowdsourcing, including product design, software development, and game production. This research demonstrates digitally networked organizations that flexibly assemble and reassemble themselves from a globally distributed online workforce to accomplish complex work.

ACM Classification Keywords
H.5.3. Information Interfaces and Presentation (e.g. HCI): Group and Organization Interfaces

Author Keywords
Crowdsourcing; expert crowd work; flash organizations

INTRODUCTION
Crowdsourcing mobilizes a massive online workforce into collectives of unprecedented scale. The dominant approach for crowdsourcing is the microtask workflow, which enables contributions at scale by modularizing and pre-specifying

actions [7, 55]. By drawing together experts [71] or amateurs [6], microtask workflows have produced remarkable success in robotic control [48], data clustering [12], galaxy labeling [54], and other goals that can be similarly pre-specified. However, goals that are open-ended and complex, for example invention, production, and engineering [42], remain largely out of reach. Open-ended and complex goals are not easily adapted to microtask workflows because it is difficult to articulate, modularize, and pre-specify all possible actions needed to achieve them [72, 81]. If crowdsourcing remains confined to only the goals so predictable that they can be entirely pre-defined using workflows, crowdsourcing's long-term applicability, scope and value will be severely limited.

In this paper, we explore an alternative crowdsourcing approach that can achieve far more open-ended and complex goals: crowds structured like *organizations*. We take inspiration from modern organizations because they regularly orchestrate large groups in pursuit of complex and open-ended goals, whether short-term like disaster response or long-term like spaceflight [8, 9, 64]. Organizations achieve this complexity through a set of formal structures — roles, teams, and hierarchies — that encode responsibilities, interdependencies and information flow without necessarily pre-specifying all actions [15, 84].

Figure 1: Flash organizations are crowds computationally structured like organizations. They enable automated hiring of expert crowd workers into role structures, and continuous reconfiguration of those structures to direct the crowd's activities toward complex goals.

Engineering: Dream Team

This project used multi-armed bandits to identify over several rounds of interaction whether teams should be flat or hierarchical, supportive or critical, etc. But we didn't know: could these multi-armed bandits actually converge fast enough to be useful?

We had a rough implementation of the multi-armed bandits, but it wasn't production ready for interacting with teams.

Engineering: Dream Team

We used a rough simulation! Assuming some roughly accurate numbers in how much each team benefited from each bandit setting, we generated teams and simulated the bandits over a few rounds.

The answer: they converged quickly enough that this might work!

(The next step: wizard of oz the interface, so we could test it "for real" without building integrating software.)

Design: Structured feed

We had a hunch that social media feeds could be much better if we had a little bit of metadata on what you're talking about. If it knew that you're posting about an episode of Westworld, or playing a game of basketball, or studying for a specific class…could it make you seem really engaging?

Like an Instagram filter for other kinds of activity: make you seem better at composition than you really are.

We sketched out a few ideas and then hired Upwork designers to create some mocks of what they might look like. (We decided it wasn't cool enough and dropped the project for the time being.)
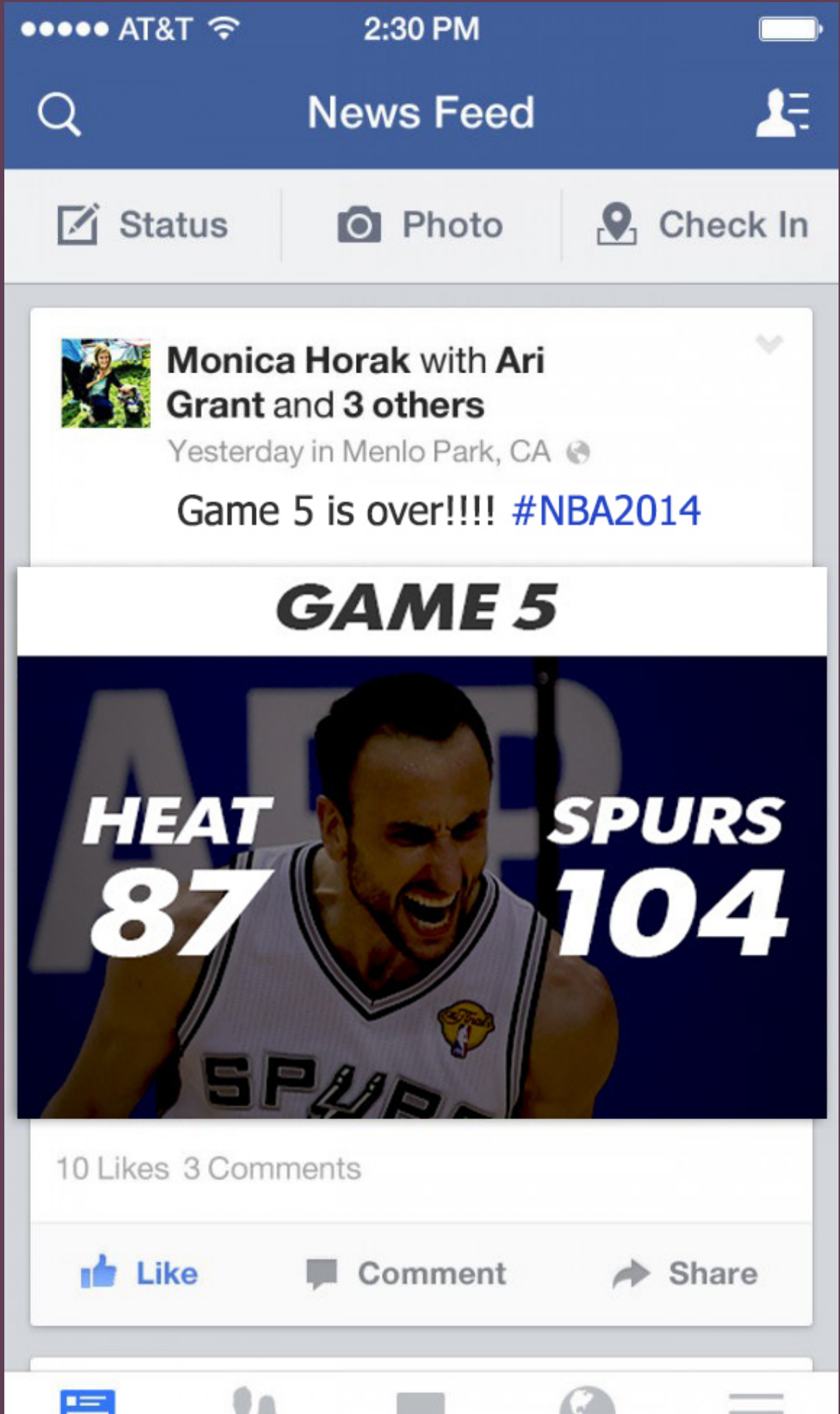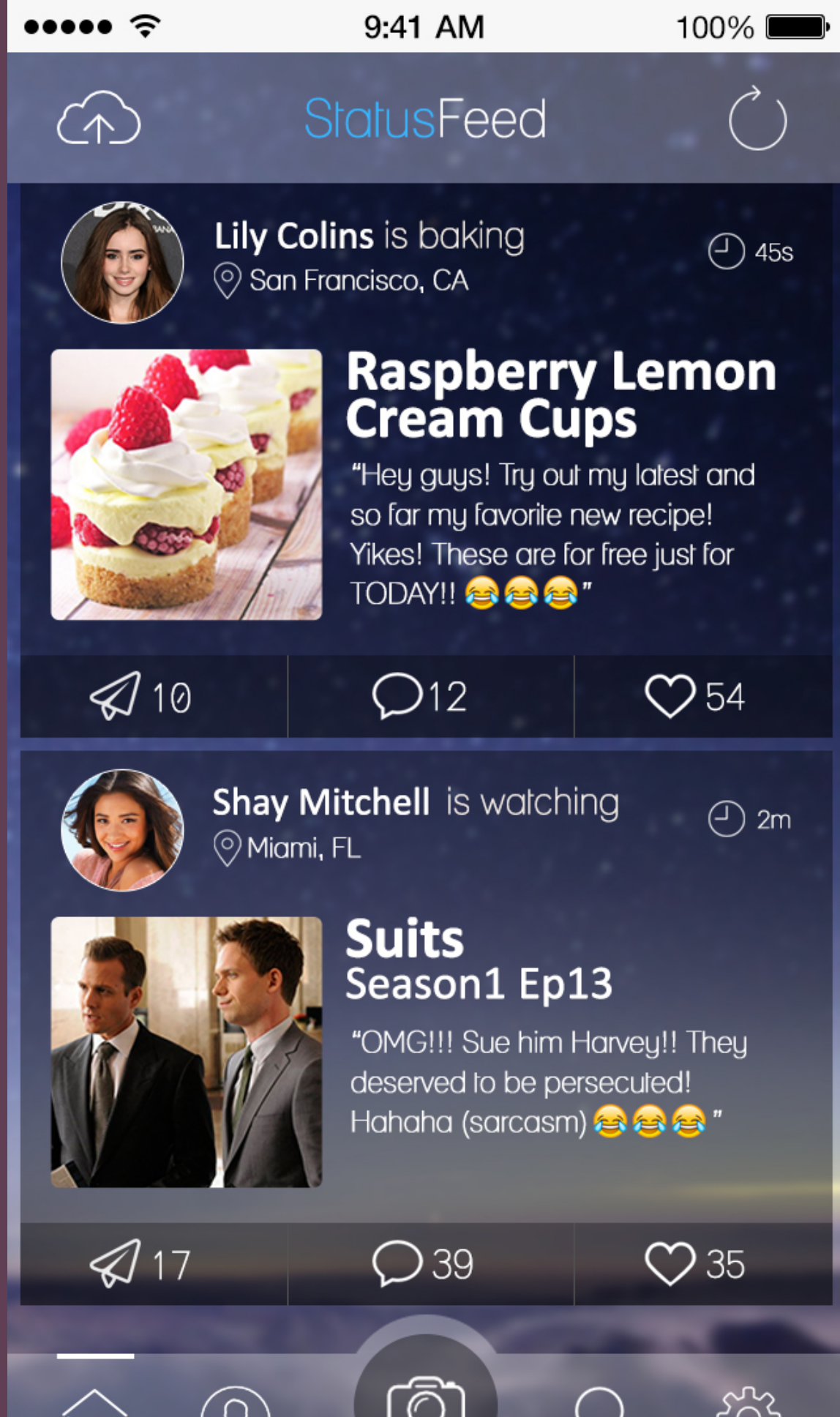
Your turn

Pair up with someone not on your project.

5min each person: describe your project's current state, the current question you're trying answer. Brainstorm together how to increase velocity.

Afterwards, we'll share out.

A reminder: the algorithm

1. Articulate the question you're answering.

2. Decide what's absolutely core to answering that question.

3. Decide what's peripheral.

4. Decide the level of fidelity that is absolutely necessary.

5. Go — but be open to reevaluating your assumptions as you go.

6. Loop with a new question.

Tips and tricks

"I'm being low velocity."

Velocity = distance / time

So, if your velocity is low, you have two options:

1. Cover more distance: habits that can get you further in the same time (e.g., "try harder", "be a better engineer")

   You're typically already maxed out on

2. Decrease the time: prototype more effectively

   WIN. Prototype more narrowly, lower your fidelity expectations (e.g., spit out

Checking email or InstaSnapFace?

This signals a lack of focus, and is a pretty certain predictor that you're in a swamp.

It means you're prototyping too broadly: you're unfocused! focus your goal. Or you're requiring too high a level of fidelity: you have unreasonable standards! lower your expectations.

Develop an internal velocity sensor, and as soon as you recognize this, apply one of the two rules.

Lowering standards: parallelism

Too often, we suffer from what's known in the literature as fixation: being certain in an idea and pursuing it to the exclusion of all else. We cannot separate ego from artifact.

Instead, to answer the question, it's often best to explore multiple approaches in parallel.

> "While the quantity group was busily churning out piles of work—and learning from their mistakes—the quality group had sat theorizing about perfection, and in the end had little more to show for their efforts than grandiose theories and a pile of dead clay."
> — Bayles and Orland, 2001

Corollary 1: pivoting

Velocity is why cutting yourself off short and pivoting to a new project can be so dangerous in research.

Typically people pivot after a week in the swamp (the "fatal flaw fallacy"), rather than iterating with high velocity out of the swamp.

I promise that the project you pivot to will have a swamp too. Learn to increase velocity and prototype your way out of the swamp faster, instead of seeking out a swampless project.

Corollary 2: technical debt

Obviously, at some point you need to make sure you're not too deep in technical debt, design debt, or writing debt.

But luckily, most people can only run their processors hot for a few hours a day. Everything I've described takes a lot out of you.

When you're out of creative cycles, spend time maturing other parts of your project that are no longer open questions. Or, sometimes we reach a phase where we pause prototyping and focus on refinement and execution for a bit.

Why is velocity so important?

Great research requires high velocity

Don't let 6-12 month paper deadlines obscure the velocity at which research needs to move in order to succeed.

If you want to achieve a high impact idea, you need to try a lot of approaches and refine and fail a lot. You want to do that as quickly as possible.

If you can prototype and learn and fail 5x as quickly as the next person, you will be able to achieve far more risky and impactful research.

Takeaways, in brief

1) The swamp is real, and it slows visible progress.

2) Velocity is a far better measure of yourself than progress, and it's something you actually have control over.

3) Achieve high velocity by being clear what question you're answering, and focusing ruthlessly on the core of that question while stripping out the periphery.

4) If you're low velocity, velocity = distance / time. Either increase distance (rarely possible) or decrease time (often possible: you're too broad or too perfectionist).

And finally…

Get into your project groups and discuss your strategy for velocity.
What's working? What can be improved?

Velocity in Research