Jelani Nelson

About me: I am an EECS faculty member at UC Berkeley, where I am a member of the Theory Group.

I am program committee chair of FOCS 2022. Submit your best work!

Current students:
- Ishaq Aden-Ali (co-advised with Peter Bartlett)
- Xin Lyu (co-advised with Avishay Tal)
- Haofeng (Fred) Zhang.

Current postdocs:
- Jan van den Brand (Simons Fellow)
- Shay Golan (Fulbright Scholar)

Former students:
- Vasileios Nakos. Postdoc, Saarland University.

Former postdocs:
- Paris Syminelakis. Research Staff Member, The Voleon Group.
- Kyle Luh. Assistant Professor, University of Colorado Boulder
- Rasmus Kyng. Assistant Professor, ETH Zurich.
- Huacheng Yu (co-hosted by Madhu Sudan). Assistant Professor, Princeton University.
- Jakub Pachocki. Researcher, OpenAI.
- Yi Li. Assistant Professor, Nanyang Technological University.

Other activities: AddisCoder, JamCoders, USVICoder, workshop on chaining, Simons program on probability, geometry, and computation in high dimensions.

• Professor @ UC Berkeley, previously Harvard (2013-2019)
• PhD from MIT, 2011
• Research keywords: Theory, big data, algorithms
Final presentations: **Weds, June 8 9:30am-11am**, Brown Institute (Gates 174). There will (most likely) be breakfast!

Research logs are due at class time each week

The best place to contact us is via Slack or the staff mailing list (cs197@cs.lists.stanford.edu)
Class Norms

- Allow thinking time before class discussions
- Have a community to discuss ideas with in class
- Instructors should be easily accessible for help; students should know what to ask
Research introduces a fundamental **new idea** into the world

CS research can create sea changes in how we build computational systems and use them; these sea changes can drive major shifts in industry.

CS research draws on many different methods — e.g., engineering, proof, design, probability, modeling — in different subfields.
How do we get to the point where we know what has been done, and why our idea is different, new, and exciting?

**Bit Flip**: How does the Related Work section support our proposed bit flip at the literature level?

**Literature Search**: How do we find/skim papers to find connections?

**Related Work**: How do we synthesize and present the literature to a reader?

We’ll be using these skills in **Assignment 2**, out today and due in 2 weeks.
Crowdsourcing is the process of making an open call for contributions to a large group of people online [7, 37]. In this paper, we focus especially on crowd work [42] (e.g., Amazon Mechanical Turk, Upwork), in which contributors are paid for their efforts. Current crowd work techniques are designed for decomposable tasks that are coordinated by workflows and algorithms [55]. These techniques allow for open-call recruitment at massive scale [67] and have achieved success in modularizable goals such as copyediting [6], real-time transcription [47], and robotics [48]. The workflows can be optimized at runtime among a predefined set of activities [16]. Some even enable collaborative, decentralized coordination instead of step-by-step instructions [46, 48]. As the area advanced, it began to make progress in achieving significantly more complex and interdependent goals [43], such as knowledge aggregation [30], writing [43, 61, 78], ideation [84, 85], clustering [12], and programming [11, 90].

One major challenge to achieving complex goals has been that microtask workflows struggle when the crowd must define new behaviors as work progresses [43, 44]. If crowd workers cannot be given plans in advance, they must form such action plans themselves [51]. However, workers do not always have the context needed to author correct new behaviors [12, 81], resulting in inconsistent or illegal changes that fall short of the intended outcome [44].

Recent work instead sought to achieve complex goals by moving from microtask workers to expert workers. Such systems now support user interface prototyping [70], question-answering and debugging for software engineers [11, 22, 50], worker management [28, 45], remote writing tasks [61], and skill training [77]. For example, flash teams demonstrated that expert workflows can achieve far more complex goals than can be accomplished using microtask workflows [70]. We in fact pilot the current study using the flash team approach, but for our purposes, we are interested in comparing and open-ended goals because these goals could not be fully decomposed a priori. We realized that flash teams, like other crowdsourcing approaches, still relied on immutable workflows akin to pre-specified procedures for virtual coordination, such as loosely coupled work. Peer production tends to be decentralized, with contributions to a large group of people online [7, 37]. [2] groups of individuals (such as teams) that support local problem-solving and interdependent work [13, 29], and (3) hierarchical structures that support the aggregation of information and broad communication of centralized decisions [15, 87]. Flash organizations computationally represent these structures, which allows them to be visualized and edited, and uses them to guide work and hire workers. Some organizational designs (e.g., holacracy) are beginning to computationally embed organizational structures, but flash organizations are the first centralized organizations that exist entirely online, with no off-line complement. Organizational theory also describes how employees and employers are typically matched through the employee’s network [23], taking on average three weeks for an organization to hire [17]. Flash organizations use call-to-labor markets to recruit interested workers on-demand, which differs dramatically from traditional organizations and requires different design choices and coordination mechanisms.

Organizational design research also provides important insight into virtual and distributed teams. Many of the features afforded by collocated work, such as information exchange [64] and shared context [14], are difficult to replicate in distributed and online environments. Challenges arise due to language and cultural barriers [62, 34], incompatible time zones [65, 68], and misaligned incentives [26, 66]. Flash organizations must design for these issues, especially because the workers will not have met before. We designed our system using best practices for virtual coordination, such as loosely coupled work structures [39, 64], online visualization [20, 27], current awareness [10, 57], and rich communication tools [64].
The Bit Flip
Recall: Novelty

If the idea is already in the world, it is not considered novel, and thus not research.

In other words, to do research, you need to achieve something that nobody else has ever done. That novel achievement is called the contribution of your research.

You’ll hear people say things like:

“This is an extremely novel contribution.”

“This work is a tad too incremental.” (Its improvement or level of creativity over the state of the art is only minimal)
OK, how do I get a novel idea?

Novel ideas rarely spring forth fully formed from a researcher’s head. They’re not cool ideas that erupt out of the void.

They’re much more often pivoted off of today’s work:

- Some constraint that exists but shouldn’t, or visa versa

- A realization that an idea has been applied in domains like X and needs to be rethought in domains like \( \sim X \) \( \text{“not } X \text{”} \)

- A recognition that others have tried this technique in users of context A, or data of up to size N, but \( \sim A \) or \( >> N \) breaks the technique. \( \text{“much greater than } N \text{”} \)

In other words, research ideas arise as a reaction to the researcher’s understanding of how people think about the problem today.
Bit flip: invert an assumption

Those examples were instances of a bit flip: an inversion of an assumption that the world has about how the world is supposed to work.

Recipe for a bit flip:

1) Articulate an assumption, often left implicit in prior work: this is the bit

2) “No, it should be this way instead:” argue for an alternative to that assumption
Recall: Turing award winning bit flips

**Bit**
We need complicated instruction sets to accommodate powerful computer processors.

**Flip**
Simple instruction sets are better since they let you compare performance, optimize, and prevent errors.

**Project**
RISC architecture

Computing was just for numerical calculations: slow, done in batches, and for teams.

Computing should be interactive, individual, and support thought.

Neural networks exist, but don’t perform very well and aren’t accurate.

We need more data and different algorithms for this to work.

Rethink domain assumptions

Remove old constraint

Old technique, new context + data

Mother of All Demos

Deep learning
<table>
<thead>
<tr>
<th>Bit</th>
<th>Flip</th>
<th>Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network behaviors are defined in hardware, statically.</td>
<td>If we define the behaviors in software, networks can become more</td>
<td>Software-defined networking</td>
</tr>
<tr>
<td></td>
<td>dynamic and more easily debuggable.</td>
<td></td>
</tr>
<tr>
<td>Code compilers should utilize smart algorithms to optimize into</td>
<td>Code compilers will find more efficient outcomes if they just do</td>
<td>STOKE</td>
</tr>
<tr>
<td>machine code.</td>
<td>Monte Carlo (random!) explorations of optimizations.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A randomized, probabilistic algorithm will be much faster, and</td>
<td>Karger’s algorithm</td>
</tr>
<tr>
<td></td>
<td>we can still prove a limited probability of an error.</td>
<td></td>
</tr>
</tbody>
</table>


Activity tracking requires custom hardware.

NLP machine learning models should read sentences word by word, so the model can see what’s before the current word.

Activity tracking requires just a standard cell phone.

NLP machine learning models should consume the entire sentence at once, so the parser can see what’s before and after.

Bit

Activity tracking requires custom hardware.

NLP machine learning models should read sentences word by word, so the model can see what’s before the current word.

Flip

Activity tracking requires just a standard cell phone.

NLP machine learning models should consume the entire sentence at once, so the parser can see what’s before and after.

Project

BERT

remove old constraint

rethink domain assumptions


Your mentor gave you a paper that is adjacent to your idea. Think of this as your nearest neighbor paper.

Your paper will be some sort of delta off of that paper.

What assumption or limitation did it have, that you’re erasing?
Literature search graph

influence

Nearest neighbor → Your project
Imagine a set of design axes. Your project should maintain position on most of them, but differentiate itself along one axis.

You can use a new technique to solve an old problem. Or solve a new problem with an old technique. If you try to do both, things get tricky...
Single paper bit flip: example

Nearest neighbor paper [Follmer et al. 2013]:

Your idea: use small mobile robots to manipulate objects

What’s the bit flip?

It might help to articulate the assumption, how you’re changing it, and what that enables

https://tangible.media.mit.edu/project/inform/
Was this what you had in mind?

Zooids [Le Goc et al., 2016]

Zooids: Building Blocks for Swarm User Interfaces

Mathieu Le Goc¹,²,³, Lawrence H. Kim³, Ali Parsaei², Jean-Daniel Fekete¹,⁴, Pierre Dragicevic¹,⁴, Sean Follmer²

¹ Inria, ² Stanford University, ³ Université Paris-Sud, ⁴ Université Paris-Saclay

https://shape.stanford.edu/research/swarm/
Literature-level bit flip

There exist many possible bit flips for a single paper:

What if we changed the size of the pins?
What if we vibrated the pins for haptic feedback?
Could we make it mobile rather than mounted on a table?

...but not every possible bit flip matters. Some are incremental.

You identify more important ideas by bit flipping across a broader literature.
Pivoting off the literature broadly, not just off a single paper, makes for a stronger argument of novelty.

Recipe:

1) Read the literature. (Which papers? Stay tuned...)

2) What assumptions underlie all of the papers? $\forall p \in \text{papers}$: ...

3) Which assumption are you changing? And why does it matter to the literature?
One paper: many possible bits

Each separating line is a possible bit flip.

Which one is best?
The broader an understanding you have of the literature and the design axes underneath it, the more effectively you can pick the right bit flip.
The broader an understanding you have of the literature and the design axes underneath it, the more effectively you can pick the right bit flip.
Literature-level bit flip

If your nearest neighbor paper assumed X and you want to do ~X, but other papers in the domain already assumed ~X with slightly different setups, it’s a more minor contribution.

Example:

You are creating a visual question answering algorithm that can look at a picture and answer a question about it. The nearest neighbor used an LSTM architecture.

You want to use a BERT architecture. That’s a paper-level bit flip.

But non-computer vision question answering algorithms already use BERT. So it’s still a contribution, just not as broad.
Ultimately...

It’s unlikely that you will find an idea that nobody has ever articulated in any context ever.

Instead, your goal is to articulate the broadest class of papers possible that your bit flip applies to. That’s considered the **scope of your contribution**.
Outcome

This all gets communicated in a Related Work section. (And to a more limited extent, in the Introduction section.)

A related work section lays out the literature in a way that the reader can understand what you're building on, and what your bit flip is relative to that prior research.

RELATED WORK

In this section, we motivate flash organizations through an integration of the crowdsourcing and organizational design research literature, and connect their design to lessons from distributed work and peer production (Table 1).

Crowdsourcing workflows

Crowdsourcing is the process of making an open call for contributions to a large group of people online [7, 37]. In this paper, we focus especially on crowd work [42] (e.g., Amazon Mechanical Turk, Upwork), in which contributors are paid for their efforts. Current crowd work techniques are designed for decomposable tasks that are coordinated by workflows and algorithms [55]. These techniques allow for open-call recruitment at massive scale [57] and have achieved success in modularizable goals such as crowd-sourcing [6], real-time transcription [47], and robotics [48]. The workflows can be optimized at runtime among a predefined set of activities [16]. Some even enable collaborative, decentralized coordination instead of step-by-step instructions [46, 86]. As the area advanced, it began to make progress in achieving significantly more complex and interdependent goals [43], such as knowledge aggregation [30], writing [43, 61, 78], ideation [84, 85], clustering [12], and programming [81, 30].

One major challenge to achieving complex goals has been that microtask workflows struggle when the crowd must define new behaviors as work progresses [43, 44]. If crowd workers cannot be given plans in advance, they must still define actions themselves [51]. However, workers do not always have the context needed to author correct new behaviors [72, 81]. resulting in inconsistent or illogical changes that fall short of the intended goals [39, 42].

Recent work instead sought to achieve complex goals by moving from microtask workers to expert workers. Such systems now support user interface prototyping [70], question-answering and defined software engineers [11, 22, 50], worker management [28, 45], remote writing tasks [61], and skill training [77]. For example, flash teams demonstrated that expert workflows can achieve far more complex goals than can be accomplished using microtask workflows [70]. We in fact piloted the current study using the flash teams approach, but the flash teams kept failing at complex and open-ended goals because these goals could not be fully decomposed a priori. We realized that flash teams, like other crowdsourcing approaches, still relied on immutable workflows akin to an assembly line. They always used the same pre-specified sequence of tasks, roles, and dependencies.

Rather than structuring crowds like assembly lines, flash organizations structure crowds like organizations. This perspective implies major design differences from flash teams. First, workers no longer rely on a workflow to know what to do; instead, a centralized hierarchy enables more flexible, de-individuated coordination without pre-specified plans for all workers’ behaviors. Second, flash teams are restricted to fixed tasks, roles, and dependencies, whereas flash organizations introduce a pull request model that enables them to fully reconfigure any organizational structure enabling open-ended adaptation that flash teams cannot achieve. Third, whereas flash teams hire the entire team at once in the beginning, flash organizations’ adaptation means the role structure changes throughout the project, requiring on-demand hiring and onboarding. Taken together, these affordances enable flash organizations to scale to much larger sizes than flash teams, and to accomplish more complex and open-ended goals. So, while flash teams’ predefined workflows enable automation and optimization, flash organizations enable open-ended adaptation.

Organizational design and distributed work

Flash organizations draw on and extend principles from organizational theory. Organizational design research theorizes how a set of customized organizational structures enable coordination [52]. These structures establish (1) roles that encode the work responsibilities of individual actors [41], (2) groupings of individuals (such as teams) that support local problem-solving and interdependent work [13, 29], and (3) hierarchies that support the aggregation of information and communication of centralized decisions [15, 87]. Flash organizations computationally represent these structures, which allows them to be visualized and edited, and used to guide work and hire workers. Some organizational designs (e.g., holacracy) are beginning to computationally embed organizational structures, but flash organizations are the first centralized organizations that exist entirely online, with no offline complement. Organizational theory also describes how employees and employers are typically matched through the employer’s network [23], taking on average three weeks for an organization to hire [17]. Flash organizations use open-calls to online labor markets to recruit interested workers on-demand, which differs dramatically from traditional organizations and requires different design choices and considerations.

Organizational design research also provides important insight into virtual and distributed teams. Many of the features afforded by collocated work, such as information exchange [64] and shared context [14], are difficult to replicate in distributed and online environments. Challenges arise due to language and cultural barriers [62, 34], incompatible time zones [65, 68], and misaligned incentives [26, 66]. Flash organizations must design for these issues, especially because the workers will not have met before. We designed our system using best practices for virtual coordination, such as loosely coupled work structures [35, 64], situational awareness [20, 27], current-state visualization [10, 57], and rich communication tools [64].

Peer production

Flash organizations also relate to peer production [3]. Peer production has produced notable successes in Wikipedia and in free and open source software. One of the main differences between flash organizations and peer production is whether idea conception, decision rights, and task execution are centralized or decentralized. Centralization, for example through a leadership hierarchy, supports tightly integrated work [15, 87]; decentralization, as in wiki software, supports more loosely coupled work. Peer production tends to be decentralized, which offers many benefits, but does not easily support integration across modules [4, 33], limiting the complexity of the resulting work [3]. Flash organizations, in contrast, use centralized structures to achieve integrated planning and coordination,
RELATED WORK

In this section, we motivate flash organizations through an integration of the crowdsourcing and organizational design research literature, and connect their design to lessons from distributed work and peer production (Table 1).

Crowdsourcing workflows

Crowdsourcing is the process of making an open call for contributions to a large group of people online [7, 37]. In this paper, we focus especially on crowd work [42] (e.g., Amazon Mechanical Turk, Upwork), in which contributors are paid for their efforts. Current crowd work techniques are designed for decomposable tasks that are coordinated by workflows and algorithms [55]. These techniques allow for open-call recruitment at massive scale [67] and have achieved success in modularizable goals such as copyediting [6], real-time transcription [47], and robotics [48]. The workflows can be optimized at runtime among a predefined set of activities [16]. Some even enable collaborative, decentralized coordination instead of step-by-step instructions [46, 86]. As the area advanced, it began to make progress in achieving significantly more complex and interdependent goals [43], such as knowledge aggregation [30], writing [43, 61, 78], ideation [84, 85], clustering [12], and programming [81, 301].

One major challenge to achieving complex goals has been that microtask workflows struggle when the crowd must define new behaviors as work progresses [43, 44]. If crowd workers cannot be given plans in advance, they must form such action plans themselves [51]. However, workers do not always have the context needed to author correct new behaviors [72, 81], resulting in inconsistent or illogical changes that fall short of the intended outcome [44].

Recent work instead sought to achieve complex goals by moving from microtask workers to expert workers. Such systems now support user interface prototyping [70], question answering and debugging for software engineers [11, 20, 50], worker management [28, 45], remote writing tasks [61], and skill training [77]. For example, flash teams demonstrated that expert workflows can achieve far more complex goals than can be accomplished using microtask workflows [70]. We in fact piloted the current study using the flash teams approach, but the flash teams kept failing at complex and open-ended goals because these goals could not be fully decomposed a priori. We realized that flash teams, like other crowdsourcing approaches, still relied on immutable workflows akin to step-by-step instructions [44].

Rather than structuring crowds like assembly lines, flash organizations structure crowds like organizations. This perspective implies major design differences from flash teams. First, workers no longer rely on a workflow to know what to do; instead, a centralized hierarchy enables more flexible, de-individualized coordination without pre-specifying all workers’ behaviors. Second, flash teams are restricted to fixed tasks, roles, and dependencies, whereas flash organizations introduce a pull request model that enables them to fully reconfigure any organizational structure enabling open-ended adaptation that flash teams cannot achieve. Third, whereas flash teams hire the entire team at once in the beginning, flash organizations’ adaptation means the role structure changes throughout the project, requiring on-demand hiring and onboarding. Taken together, these affordances enable flash organizations to scale to much larger sizes than flash teams, and to accomplish more complex and open-ended goals. So, while flash teams’ predefined workflows enable automation and optimization, flash organizations enable open-ended adaptation.

Organizational design and distributed work

Flash organizations draw on and extend principles from organizational theory. Organizational design research theorizes how a set of customized organizational structures enable coordination [52]. These structures establish (1) roles that encode the work responsibilities of individual actors [41], (2) groupings of individuals (such as teams) that support local problem-solving and interdependent work [13, 29], and (3) hierarchies that support the aggregation of information and broad communication of centralized decisions [15, 87]. Flash organizations computationally represent these structures, which allows them to be visualized and edited, and uses them to guide work and hire workers. Some organizational designs (e.g., holacracy) are beginning to computationally embed organizational structures, but flash organizations are the first centralized organizations that exist entirely online, with no offline complement. Organizational theory also describes how employees and employers are typically matched through the employee’s network [23], taking on average three weeks for an organization to hire [17]. Flash organizations use open-calls to online labor markets to recruit interested workers on-demand, which differs dramatically from traditional organizations and requires different design choices and coordination mechanisms.

Organizational design research also provides important insight into virtual and distributed teams. Many of the features afforded by collocated work, such as information exchange [64] and shared context [14], are difficult to replicate in distributed and online environments. Challenges arise due to language and cultural barriers [62, 34], incompatible time zones [65, 68], and misaligned incentives [26, 66]. Flash organizations must design for these issues, especially because the workers will not have met before. We designed our system using best practices for virtual coordination, such as loosely coupled work structures [35, 64], situational awareness [20, 27], current state visualization [10, 57], and rich communication tools [64].

Peer production

Flash organizations also relate to peer production [3]. Peer production has produced notable successes in Wikipedia and in free and open source software. One of the main differences between flash organizations and peer production is whether idea conception, decision rights, and task execution are centralized or decentralized. Centralization, for example through a leadership hierarchy, supports tightly integrated work [15, 87]; decentralization, as in wiki software, supports more loosely coupled work. Peer production tends to be decentralized, which offers many benefits, but does not easily support integration across modules [4, 33], limiting the complexity of the resulting work [3]. Flash organizations, in contrast, use centralized structures to achieve integrated planning and coordination,
Performing a literature search

“An hour in the library saves you a year at the keyboard.”
Goal: build the literature graph

While I don’t literally draw out the graph, building up that understanding in my head and visualizing a few different axes is key in identifying the right bit flip.

So how do we get there?
Step one: nearest neighbor

Start with a seed paper that is the closest in the design space to yours. This is your nearest neighbor paper.

Read this paper in depth. Understand it.

(Your nearest neighbor paper may not be a great paper! Often great ideas are adjacent to a near miss.)
influence

Nearest neighbor → Your project
Step two: expand your horizon

What are the 3–5 most important citations to that nearest neighbor?

Look at the papers that it cited visibly and carefully in the Introduction and Related Work

Look at which papers it is arguing a bit flip from

Are those most important citations staying in the neighborhood of your topic, or going somewhere else? Keep the ones that are staying in the neighborhood.
How to expand the horizon

Backward influence: influential citations in the papers that you’ve read

Tools: reading

Forward influence: papers citing the ones that you’ve read

Tools: Google Scholar’s “Cited By”, Semantic Scholar’s “highly influenced” designation

Relatedness: contemporaneous but not citing

Tools: Google Scholar’s “Related articles”
influence

Nearest neighbor → Your project
First expansion
Second expansion
Final set
Filtering your horizon

Not all papers achieve the same level of quality. Especially on white paper archives such as arXiv.org, quality can be variable.

How do I know what to read and what to ignore?

Published at a reputable venue: ask your mentor/course staff for the reputable venues in the field of your project, and stick to those venues. (Or ask if you find a relevant paper outside of those venues and want a gut check.)

Published from a reputable institution or author: again, you might ask your mentor whether an institution you’ve never heard of is reputable.
The halting problem

How do you know when to stop reading? When do you have enough confidence that your idea is the right contribution to pursue?

What if you’ve missed something?
Asymptoting

Keep track of **how much you’re learning about the design axes** as you consume the additional papers. Typically, you are learning the most at the very beginning, and the amount per paper starts going down after five papers or so.

A PhD student often asymptotes after 25–35 papers.

For this class, we’ll go with 15 for now.
Reading a paper for a literature search
Temptation: understand everything.

Typically, when we come to a paper, we want to understand everything about it. We stop and reread any point we don’t get.

This can take an hour or two per paper for a new researcher.

This strategy can be useful at the beginning, but it is actively harmful in constructing a related work section.

⚠ Not recommended!
Understand the main point

Instead, articulate to yourself: **what is the main point** that this paper is making?

Then, focus your reading and effort most closely on the parts of the paper that are supporting or evaluating that point.

It’s OK not to understand every sentence in the paper.

**Your goal isn’t to understand the paper — it’s to understand the literature,** what works and what doesn’t (and why!), and the bits that are available to flip.
Tip: Read with Purpose

Papers in your literature search typically fall into 2 categories:

- Work that informs how you **implement** your bit flip
- Work that provides additional **context**/contrast

You will undoubtedly reread papers in the first category in detail when you begin your project work, but now is not that time.

Remember Assignment 1:

Outlining a Paper

Your outline should recompress the paper back following structure of one paragraph per bullet:

- **Title**: What paper did you read?
- **Problem**: What problem is it solving? Why?
- **Assumption in prior work**: What was the assumption inadequate?
- **Insight**: What is the novel idea that this paper advances?
Tip: Skim Effectively

Understand the Problem, the Assumption, and the Insight by reading **abstract and introduction in great depth** understanding **terminology** from abstract and introduction **continue search** by checking the related work section

Use a Citation Manager:

Google spreadsheet list of links, Overleaf list of BibTeX snippets, or Zotero/Mendeley that explicitly does both
Writing the Related Work
Step 1: affinity mapping

Put each paper onto a post-it note

Place the post-it notes onto a whiteboard or wall, placing similar ideas close to each other.

Group by whatever makes sense to you.
Step 2: regrouping

Typically, these first groups represent topics (nouns). This isn’t wrong, but it’s not the most helpful in writing a Related Work section.

So, instead, aim for each group to have a shared thesis behind it, not just a noun. Regroup your post-its around shared theses.

### Not good:

- Rigging

### Better:

To automatically construct rigs, past research either has done rig transfer methods, or rig synthesis methods.
Step 3: bit flip

Now that you have a thesis that encompasses a group of papers, how does your project differ? What assumptions are they making? What design axes do you change?

**Bit:**
To automatically construct rigs, past research either has done rig transfer methods, or rig synthesis methods.

**Bit flip:**
When working with 2D character creation data, rig transfer doesn’t work since the data is too diverse and rig synthesis doesn’t work since it assumes 3D data. Thus, we develop a new method that is purely geometric.
Step 3: outlining

Each thesis then becomes the topic sentence of a paragraph of the Related Work section, and each bit flip ends the paragraph.

To automatically construct **rigs**, past research either has done rig transfer methods, or rig synthesis methods.

**Mix & match assembly based modeling** helps novice users create artifacts they might otherwise lack the skill to.

Many **deformation** techniques for 2D and 3D geometry need user input.

Existing methods don't work (explain why)

These systems don't adapt to changes, we built a system that does

Our system enables deformations without user input
The temptation when writing is to list all the related work under the topic. Don’t do this. Instead, start with the thesis, and use the paragraph to prove the thesis.

Projects have used runtime feedback to help programmers debug projects. The Arsinine project highlighted variables as they are being accessed [4]. Achilles produced a summary report on the command line after execution completed [7]. More recently, Arbuckle played annoying sounds from the computer speakers whenever a watched variable was accessed [17].

Runtime feedback of memory accesses gives programmers an intuitive sense of whether the program’s behavior match their intuitions. Highlighting variables as they are accessed provides a rough understanding of the most-used items [4]. Complementing this report with a summary afterwards helps support this sensemaking [7]. Particularly worrisome program behaviors are more likely to be noticed when complemented by audio feedback [17].
When putting in projects, don’t just list them: describe how they related to the thesis of the paragraph.

Runtime feedback of memory accesses gives programmers an intuitive sense of whether the program’s behavior match their intuitions. Highlighting variables as they are accessed provides a rough understanding of the most-used items [4]. Complementing this report with a summary afterwards helps support this sensemaking [7]. Particularly worrisome program behaviors are more likely to be noticed when complemented by audio feedback [17].

Finally, flip the bit: how does your project differ from the projects you just cited?

Where this prior work suggests that programmers make fewer errors when given runtime feedback of memory accesses, this project demonstrates that runtime feedback of function call graphs help reduce errors as well. We draw on techniques from the prior literature such as highlighting and post-hoc summaries, and extend these with a visualization algorithm for function calls.
Assignment 2

Perform a literature search for your project

Start from your nearest neighbor paper(s)

- Expand your literature search to ~15 papers
- Affinity diagramming
- Articulating a bit flip
- Writing a 600 word Related Work section for your final paper in Overleaf with proper citations

Due: 2 weeks from now before lecture
Your To-dos

1. Assignment 2 (2 weeks)
   - Read/skim 15 papers
   - Affinity map them
   - Write a related works section

2. Weekly log

3. Meetings with Jingyi & Michelle start tomorrow!

Exit ticket: http://tiny.cc/cs197-week2
Questions?

What does the “bit flip” look like for a PhD in CS?

How to use Overleaf? Zotero/citation manager?

Is there always a bit flip?

Exit ticket: http://tiny.cc/cs197-week2
Computer Science Research

Slide content shareable under a Creative Commons Attribution-NonCommercial 4.0 International License.