A) Full configurations for ‘b’ sentences attached separately.

The following paragraph fills in the missing cells for LC-parsing (left, right, and center-branching) and center-branching for top-down. The next paragraphs all just talk about the ‘c’ sentences and justify how certain parsers wouldn’t have any noteworthy configurations that impose heavy memory loads.

Regarding the 3b sentences, I don’t know if I’m straying too far from the overall material by giving this answer, but I’d honestly argue that center-branching structures are in essence right-branching structures. “The actor the boy met won” is arguably a case of that-ellipsis where the phrase can be understood as “The actor that the boy met won”. Because of this, when I did the top-down parse for 3b to fill the missing value in the table, I honestly didn’t notice much of a difference in terms of computational difficulty. In terms of computational difficulty for the LC-parser, I don’t think I noticed an increase in difficulty when doing the tables for 1-3b using left-corner parsing.

For 1c, I’d argue there isn’t really any configuration that has noticeably more computational difficulty for bottom-up and LC parsing. Bottom-up parsers can just reduce the configuration at (NP POSS N, ‘s baby won) to (NP, ‘s baby won) and parse another possessive structure without increasing memory loads. LC parsing can simply LC-PREDICT a possessive from (NP, ‘s boss’s baby won), and once that’s done, LC parsing can just LC-predict the second possessive branch further on. In essence, there’s essentially no configuration differences here between 1b and 1c for these two parsers. However, for top-down parsing, predicting completely is significantly more expensive because it needs to predict that the innermost possessive’s NP is a series of nested possessives, leading to the expensive prediction of (NP POSS N POSS N VP, Mary’s boss’s baby won).

For 2c, bottom-up parsing would have to parse an entire right-branching structure to fully be able to reduce it. As such, before being able to reduce using S → NP VP, this configuration from 2c would be particularly expensive due to having to use shift transitions for the entire structure:
(NP V D N THAT V D N THAT V D N, epsilon). Everything after the first V needs to reduced into a single VP, but that entire configuration can’t be reduced until the end of the sentence because of the embedded that-clauses. The VP within a that-clause needs to be fully parsed before the D N SRC rule can be applied, but the embedded VPs also have NP → D N SRCs that need to be fully parsed first before being able to reduce them. 2c doesn’t pose a problem for top-down and LC-parsing because the use of predicts (LC-predicts) helps shave off the already-parsed Ds and Ns for NP → D N SRC while predicting for SRC → THAT VP helps shave off the memory load of the that’s.

For 3c, I’d treat it like a right-branching structure for the reasons explained above. As such, for a bottom-up parser, which is already going to have a high memory load from having to shift all of “the actor the boy met” into memory before being able to reduce it to the matrix NP in 3b, it’ll have an even higher memory load, with the configuration (D N D N D N V V, won) before being able to reduce eventually to (NP, won), (NP VP, epsilon), and (S, epsilon). Treating the ORC as a right-branching structure, top-down and LC parsing will have an identical memory load to 3b simply because (D N, 2) can be LC-predicted to be NP → D N ORC, clearing the D and N from memory and helping manage memory usage; the same pattern repeats for the ORC-structure embedded in “the boy the baby saw”, allowing efficient memory use by predicting NP → D N ORC again and continuing the parse.

B) Arc-standard parsing poses a negligible memory usage increase. Mostly, it involves a few extra steps for canceling, but LC-predict still helps keep down memory usage overall, making almost no difference in memory usage, if at all.

i)

0) ---	---	(S-bar, 0)
1) SHIFT	D → the	(D S-bar, 1)
2) SHIFT	N → actor	(D N S-bar, 2)
3) LC-PREDICT	NP → D N ORC	(ORC-bar NP S-bar, 2)
4) SHIFT	D → the	(D ORC-bar NP S-bar, 3)
5) LC-PREDICT	NP → D N	(N-bar NP ORC-bar NP S-bar, 3)
6) MATCH	N → boy	(NP ORC-bar NP S-bar, 4)
7) LC-PREDICT	ORC → NP V	(V-bar ORC ORC-bar NP S-bar, 4)
8) MATCH	V → met	(ORC ORC-bar NP S-bar, 5)
9) CANCEL	---	(NP S-bar, 5)
10) LC-PREDICT	S → NP VP	(VP-bar S S-bar, 5)
11) LC-PREDICT	VP → V	(V-bar VP VP-bar S S-bar, 5)
12) MATCH	V → won	(VP VP-bar S S-bar, 6)
13) CANCEL	---	(S S-bar, 6)
14) CANCEL	---	(6, 6)

ii) 1b) (VP-bar S S-bar, won); note that multiple possible configs here have 3 max elements on the stack, so not much difference this time around out of coincidence.
2b) (VP-bar SRC SRC-bar, 5); there's again a bunch of ties here, as far as I can see mostly everything goes to 3 terms on the left side of the configuration at most.
3b) (V-bar ORC ORC-bar NP S-bar, 4)
1c) (VP-bar S S-bar, won); note that because LC-parsing is good at reducing branching structures, there's essentially no differences from 1b. Also again, lots of ties.
2c) (VP-bar SRC SRC-bar, 5); again, same reason as above. Also lots of ties again.
3c) Considering the aforementioned reasoning that ORC structures are essentially that-elipsis, constituting right-branching structures, 3c is completely ungrammatical. If forced to parse it with an LC-parser, I'd imagine that given LC-parsing is good at reducing structures to keep down memory usage, it'd be the same as 3b.

C) The information here points to hypothesis 2 being the correct hypothesis.

With the knowledge that the Martians use bottom-up parsing and have a memory limitation on how many non terminal symbols can be processed before reduction can occur, 5b shows that at the very least, there is evidence that Martians cannot store 8 or more non-terminal symbols in memory. Because they use bottom-up parsing, the entire VP "met the boy that saw the actor" needs to be analyzed with individual shifts before reduction since the rule NP → (D) N (PP) (SRC) (ORC) does not allow NPs to be recursively reduced while bottom-up parsing is still going on. The aforementioned VP is parsed as (V D N THAT V D N) because the right-most non-terminal elements need to be able to be fully reduced before further reduction can occur.

Now, looking at 6c, both hypotheses 1 and 2 need an already-parsed S in the new VP rule. Since the Martians use bottom-up parsing, they need to continue shifting elements from the sentence until they can properly reduce the non-terminal elements into an embedded S. Using hypothesis 1 to parse 6c, the

configuration prior to beginning any possible reductions would be (NP SAID ADV NP SAID ADV NP V, epsilon). Again, because we're using bottom-up parsing, none of this can reduced until the final V "won" is shifted. At that point, the V reduces to VP, the rightmost NP VP reduce to S, and so on. However, from the evidence in 5b, hypothesis 1 would predict 6c to be ungrammatical since 6c generates 8 non-terminal symbols under hypothesis 1. However, hypothesis 2 does not have this issue. The sentence reduces SAID ADV pairs into Xs while parsing the sentence. This reduces memory load, and the largest configuration ends up being (NP X NP X NP V, epsilon), with reductions VP → V, S → NP VP, VP → X S, and so on. Hypothesis 2 correctly predicts 6c to be grammatical. It also correctly predicts 6d to be ungrammatical, with the most memory-heavy configuration being (NP X NP X NP X NP V, epsilon), which is already predicted to be ungrammatical by the evidence presented by 5b.

D) Trees attached separately.

E) Correct bottom-up parse:

#	Type of transition	Rule used	Configuration
0	---	---	(epsilon,0)
1	SHIFT	WHILE → while	(WHILE, 1)
2	SHIFT	NP → John	(WHILE NP, 2)
3	SHIFT	V → watched	(WHILE NP V, 3)
4	REDUCE	VP → V	(WHILE NP VP, 3)
5	REDUCE	S → NP VP	(WHILE S, 3)
6	SHIFT	D → the	(WHILE S D, 4)
7	SHIFT	N → baby	(WHILE S D N, 5)
8	SHIFT	THAT → that	(WHILE S D N THAT, 6)
9	SHIFT	V → won	(WHILE S D N THAT V, 7)
10	REDUCE	VP → V	(WHILE S D N THAT VP, 7)
11	REDUCE	SRC → THAT VP	(WHILE S D N SRC, 7)
12	REDUCE	NP → D N SRC	(WHILE S NP, 7)
13	SHIFT	V → cried	(WHILE S NP V, epsilon)
14	REDUCE	VP → V	(WHILE S NP VP, epsilon)
15	REDUCE	S → NP VP	(WHILE S S, epsilon)
16	REDUCE	S → WHILE S S	(S, epsilon)

Dead-end bottom-up parse (point at which the parse starts going the wrong way in bold)

…	…	…	…
3	**SHIFT**	**V → watched**	**(WHILE NP V, 3)**
---	---	---	Next three "bad" configurations:
**4*	*SHIFT*	*D → the*	**(WHILE NP V D, 4)*

*5	*SHIFT*	*N → baby*	*(WHILE NP V D N, 5)*
*6	*SHIFT*	*THAT → that*	*(WHILE NP V D N THAT, 6)*

The bottom-parser is going on an incorrect parse thinking that V D N THAT … will eventually reduce into a VP.

F) Correct top-down parse:

#	Type of transition	Rule used	Configuration
0	---	---	(S, 0)
1	PREDICT	S → WHILE S S	(WHILE S S, 0)
2	MATCH	WHILE → while	(S S, 1)
3	PREDICT	S → NP VP	(NP VP S, 1)
4	MATCH	NP → John	(VP S, 2)
5	PREDICT	VP → V	(V S, 2)
6	MATCH	V → watched	(S, 3)
7	PREDICT	S → NP VP	(NP VP, 3)
8	PREDICT	NP → D N SRC	(D N SRC VP, 3)
9	MATCH	D → the	(N SRC VP, 4)
10	MATCH	N → baby	(SRC VP, 5)
11	PREDICT	SRC → THAT VP	(THAT VP VP, 5)
12	MATCH	THAT → that	(VP VP, 6)
13	PREDICT	VP → V	(V VP, 6)
14	MATCH	V → won	(VP, 7)
15	PREDICT	VP → V	(V, 7)
16	MATCH	V → cried	(epsilon, epsilon)

Dead-end top-down parse (point at which the parse starts going the wrong way in bold)

…	…	…	…
4	**MATCH**	**NP → John**	**(VP S, 2)**
---	---	---	Next three “bad” configurations:
*5	*PREDICT*	*VP → V NP*	*(V NP S, 2)*
*6	*MATCH*	*V → watched*	*(NP S, 3)*
*7	*PREDICT*	*NP → D N SRC*	*(D N SRC S, 3)*

The top-down parser incorrectly predicts “the baby that won” as an object of the verb “watched”.

G) To determine whether humans use bottom-up or top-down parsing based on their eye-movements while reading (7), check whether their eyes return to “John” or “watched”. If they return to “John”, they are using top-down parsing because they initially predicted VP → V NP instead of VP → V after matching NP → John. If they return to “watched”, they are using bottom-up parsing because they initially assumed “watched” couldn’t be reduced to a VP and tried to continue shifting elements after “watched”, thinking “watched the baby that won” would be reduced later to a VP.